MASTER EN SCIENCES INFORMATIQUES

Intégration de schémas conceptuels ERA

Célis, Marc; Deudon, Eric

Award date:
1988

Awarding institution:
Universite de Namur

Facultés universitaires N.D. de la Paix Namur

Institut d'informatique

Intégration de schémas conceptuels ERA

M.Célis & E.Deudon

TOME 1

Promoteur: F.Bodart

Mémoire présenté en vue de l'obtention
du grade de
licencié et maître en informatique

Année académique 1987-1988

Abstract

The subject of this dissertation is the building of a method and programs aiming at the integration of several database schemes described in the ERA data model.

The first part introduces a method that is based on two stages; the first of which is the detection of structural mappings between two schemes. A precise definition of each kind of mapping is given in the frame of the ERA model. The second stage concerns algorithms for integration of the detected mappings.

The second part of this work is devoted to the implementation of the first stage of the method. Because the finding of structural mappings relies strongly on semantics, it is not practically manageable without a powerfull user interface. So we were induced to do some reflexions on the design of those interfaces.

Résumé

Le sujet de ce mémoire est la construction de méthodes et programmes en vue de l'intégration de schémas de bases de données exprimés dans le modèle ERA.

La première partie introduit une méthode basée sur deux étapes dont la première est la détection de correspondances inter-schémas. Une définition précise de chaque type de correspondance est donnée dans le cadre du modèle ERA.
La seconde étape décrit des algorithmes d'intégration des correspondances détectées.

La seconde partie de ce travail est consacrée à l'implémentation de la première étape de la méthode. Comme la recherche des correspondances est fortement basée sur la sémantique, il est nécessaire de faire appel à un utilisateur humain. Vu la complexité du problème, la tâche de ce dernier est grandement facilitée par l'utilisation d'une interface graphique. Nous avons donc été amenés à faire quelques réflexions sur le développement de ce genre d'interfaces.

Nous tenons à exprimer nos plus sincères remerciements à l'égard de toutes les personnes qui ont, de près ou de loin, contribué à la réalisation de ce travail.

Nous remercions plus particulièrement:

Monsieur François Bodart, promoteur de ce mémoire, qui nous à guidé et conseillé au cours de cette année académique.

Monsieur Stefano Spaccapietra et toute l'équipe BADINE, pour leur accueil chaleureux dans ce merveilleux pays qu'est la Bourgogne.

Monsieur Benoît Sacré, pour son aide dans la construction d'une architecture interactive.

Madame Thérèse Collet-Petitjean pour ses conseils sur l'élaboration des schémas de la dynamique.

Toutes les personnes de notre entourage qui nous ont encouragé et soutenu durant ce cycle d'études.

Eric Deudon
Marc Célis

TABLE DES MATIERES

PREMIERE PARTIE :
Méthode d'intégration .

Chapitre I.4: TYPOLOGIE RELATIVE AU MODELE ERA

Chapitre I.5: MAPPING STRUCTURELS

Chapitre I.6: COMPARAISON: INTEGRATION DE VUES / INTEGRATION DE BD

Chapitre I.7:COHERENCE D'UN RESEAU DE CORRESPONDANCES

Chapitre I.8: ALGORITHMES D' INTEGRATION POUR BASE DE DONNEES EXISTANTES

SECONDE PARTIE :

Implémentation de la détection des correspondances.

BIBLIOGRAPHIE

ANNEXES

PREMIERE PARTIE:

METHODE D'INTEGRATION.

Chapitre I.1:INTRODUCTION

I.1.1 Définitions importantes

I.1.2 Objectifs

I.1.3 Difficultés

I.1.4 Principes généraux de l'intégration

I.1.5 Méthode générale d'intégration

I.1. INTRODUCTION

Dans ce premier chapitre, le lecteur trouvera, après quelques définitions importantes, un exposé général des objectifs et des difficultés de l'intégration. Nous y exposerons également les grands principes d'une méthode d'intégration avant d'aborder, dans le second chapitre , les spécificités de différentes méthodes.

I.1.1 Définitions importantes

=> schéma conceptuel: description des informations qu'il conviendra de représenter dans la future base de données pour satisfaire les besoins exprimés dans l'organisation. Cette description est faite en termes des concepts d'un modèle de structuration des informations.(*)

=> structure: construction réalisée à partir des concepts du modèle de structuration des informations, modélisant au sein d'un schéma conceptuel, un objet du monde réel.

=> population: extension des occurrences d'une structure.

=> spécifications incohérentes: si elles permettent l'existence simultanée de plusieurs instanciations différentes de la modélisation d'un même objet du monde réel .

=> intégration: c'est le remplacement dans le système d'information d'une collection de schémas conceptuels, par un schéma intégré possédant les deux propriétés suivantes:

a) la **complétude**, c'est-à-dire que tout type d'information nécessaire à un utilisateur d'un schéma doit se retrouver dans le schéma conceptuel intégré.

(*)
Cette définition est identique à celle trouvée dans [Bodart, Pigneur 83] mais dont la portée est limitée à une BD et non plus au système d'information dans son intégralité, c'est-à-dire comprenant les traitements associés aux informations.

b) la **minimalité** (ou non-redondance), c'est à dire que tout type d'information ne doit être représenté qu'une seule fois dans le schéma conceptuel. Ce qui en pratique entrainera un gain de place. De plus, si une information est représentée une et une seule fois, cela supprime le risque d'incohérence pour cette information.

I.1.2 Objectifs

I.1.2.1 origines du problème:

On peut donner deux raisons à la nécessité d'intégrer, c'est à dire au fait que les organisations travaillent souvent sur une collection de BD modélisant le même domaine du réel quand elles devraient utiliser une seule BD globale.

a) La complexité de BD importantes rend obligatoire le travail de conception en équipes distinctes, plus ou moins indépendantes. Chacune de ces équipes travaille sur une partie de ce qui devrait être un schéma global mais qui donnera en réalité une collection de **schémas conceptuels**.

b) Des groupes d'utilisateurs ayant chacun développé une BD indépendemment, peuvent après un certain temps décider de fusionner ces dernières. Nous parlerons ici d'intégration de **BD existantes**.

I.1.2.2 but de l'intégration

Le but principal est de centraliser toutes les informations dans une seule BD globale de façon à réaliser les objectifs de minimalité et de complétude vus ci-dessus en occasionant le moins de changement possible pour les utilisateurs et les concepteurs.

I.1.3 Difficultés

Les principales difficultés à résoudre proviennent de **différences** structurelles et sémantiques entre les schémas à fusionner.
Les causes de ces différences sont variées:

a) **perspectives différentes:** différents groupes d'utilisateurs adoptent leur propre point de vue dans la perception d'un même objet. Seront modélisées dans le schéma conceptuel les propriétés de l'objet interessant l'organisation dont fait partie le concepteur.

b) **équivalence entre structures:** plusieurs structures peuvent modéliser le même domaine d'application de manière équivalente. C'est-à dire que toute information représentée dans une structure l'est dans sa ou ses structures équivalentes. Il faut remarquer que les modèles les plus riches permettent une plus grande variété dans la modélisation d'une même situation.

c) **erreur de conception:** des choix incorrects sur les cardinalités, les types peuvent conduire à un schéma conceptuel erroné.

Une typologie des différences induites par ces causes sera développée en détail dans le chapitre I.3.

I.1.4 Principes généraux de l'intégration

L'intégration inclut la résolution de ces conflits entre vues différentes d'une même réalité.
En général, toutes les méthodes d'intégration comprennent les deux phases suivantes:

a) **étude des correspondances inter-schémas**

=> Une **correspondance** entre deux structures de deux schémas est l'expression de la volonté de l'utilisateur d'intégrer ces deux structures.
En général, cette volonté est guidée par le fait que les structures ci-dessus modélisent des classes d'objets du monde réel de même nature.

La description précise des correspondances, c'est à dire, les critères permettant leur détection, sera détaillée dans le chapitre I.4.
Cette phase n'est pas totalement automatisable, car elle fait appel à la sémantique des schémas.
Illustrons ce fait par un exemple:
soient deux schémas conceptuels à intégrer V1 et V2 composées chacuns d'une entité posédant des attributs.

V1: **automobile**	V2: **véhicule**
attr1	**attr2** **attr3**

aucun programme totalement automatique ne saurait deviner qu'automobile est un sous-type, ou synonyme dans certains cas, de véhicule.
Néanmoins, une approche interactive sera étudiée dans la seconde partie de ce travail.

b) **intégration proprement dite** sur la base des schémas des concepteurs locaux et des correspondances inter-schémas.

Le processus d'intégration construit pour un réseau de correspondances entre deux ou plusieurs structures, une nouvelle structure placée dans le schéma intégré et telle qu'elle puisse modéliser

tous les objets du monde réel qui étaient modélisés par l'ensemble de toutes les structures en correspondance.

Cette étape peut être automatisée pour autant que l'on dispose d'une liste complète et correcte des correspondances entre les schémas concernés.
Pour chaque type de correspondance, il faut appliquer une règle d'intégration lui étant spécifique.

I.1.5 Méthode générale d'intégration

Les considérations précédentes nous amènent à détailler le processus d'intégration sous la forme de la figure 1.1

- modélisation

Au moyen des concepts d'un modèle de structuration des traitements. Dans le cadre de ce travail nous utiliserons le modèle ERA [Bodart, Pigneur 87].

- Comparaison des schéma

Les schémas sont analysés et comparés pour déterminer les correspondances entre concepts et détecter les conflits possibles.

- Intégration

Résolution des conflits et intégration proprement dite.

- Mapping opérationnel

=> Le **mapping opérationnel** est l'ensemble des règles qui sélectionnent les parties du schéma global ainsi que les occurrences de manière a satisfaire une requête exprimée sur un schéma externe.
En clair cela donne à chaque utilisateur, ce à quoi il a droit sans qu'il doive modifier les programmes d'accès à son schéma externe.

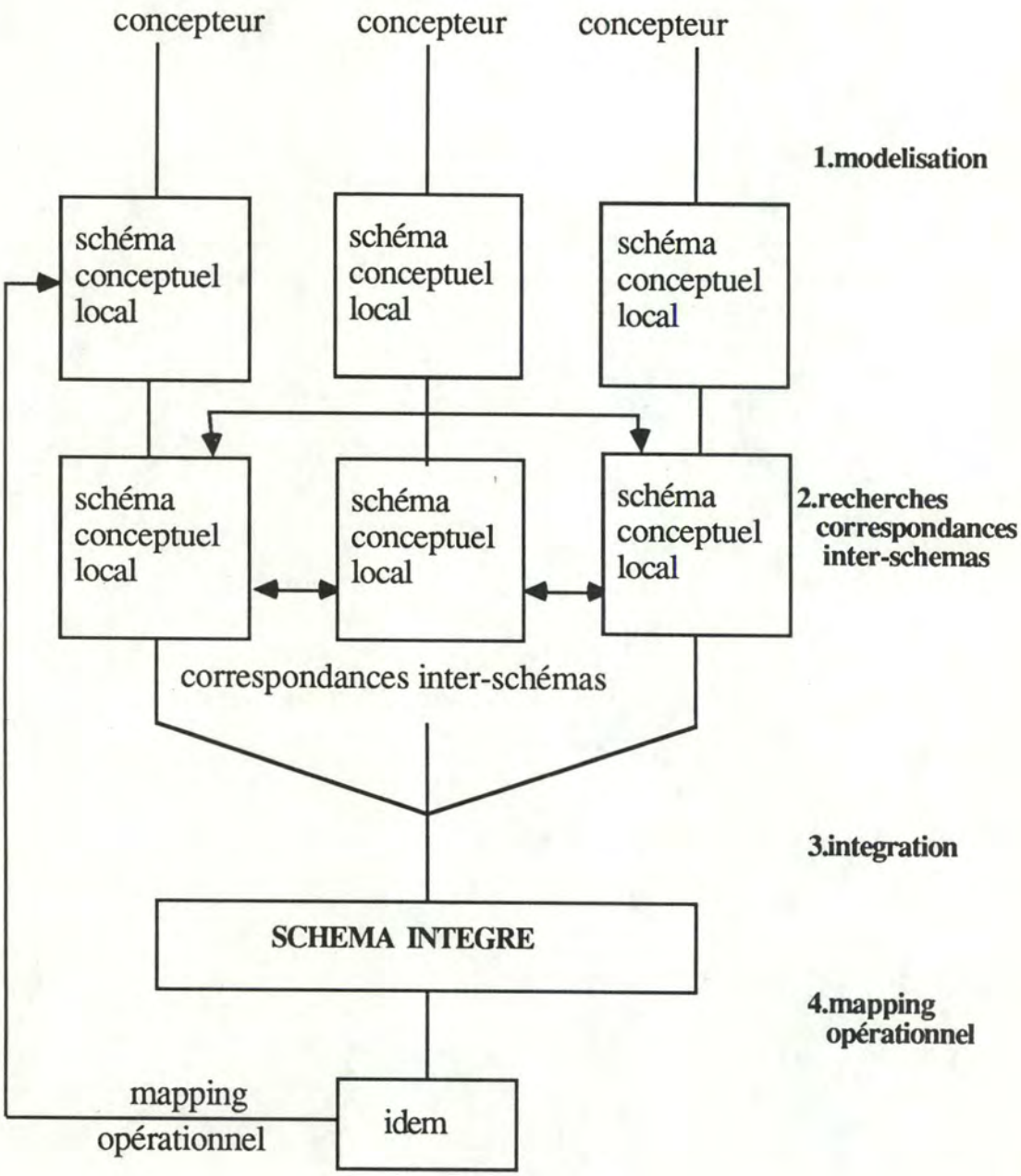


figure 1.1

Démarche générale du processus d'intégration.

A ce squelette de méthode, nous pourrions ajouter une étape de **préintégration** consistant à choisir les schémas à intégrer, leur ordre d'intégration ainsi que la possible assignation de préférences à des schémas entiers ou portions de schémas .
Exemple [Batini, Lenzerini, Navathe 86] :

Donner la préférence à des applications financières par rapport à des applications de production est un exemple de politique d'intégration qui pourrait être décidé par la direction.

Chapitre I.2: PARTICULARITE DES DIFFERENTES METHODES D'INTEGRATION

I.2.1 Le modèle de données utilisé

I.2.2 L'optique de travail

I.2.3 Les résultats obtenus

I.2.4 Le degré d'interactivité

I.2.5 Les stratégies d'intégration

I.2. PARTICULARITES DES DIFFERENTES METHODES D'INTEGRATION

Si dans l'ensemble, les méthodes d'intégration existantes respectent le plan vu précédemment, elles s'opposent par un certain nombre de points. Les 3 méthodes reprises dans les tableaux ci-dessous sont basées sur des schémas décrits sous forme ERA. Cette étude comparative a été extraite de [Grison 87].

I. 2.1 Le modèle de données utilisé

Dans le cadre de ce mémoire nous travaillerons sur des schémas exprimés dans le modèle **ERA** [Bodart, Pigneur 87].
Le tableau 1 compare les différentes méthodes sur ce point.

I. 2.2 L'optique de travail

Certaines méthodes sont applicables au moment de l'analyse conceptuelle [Batini, Lenzerini 82] alors que d'autres sont orientées vers la fusion de bases de données déjà existantes [Mannino, Effelsberg, 84].

I.2.2.1 Approche "intégration de schémas conceptuels"

Elle produit une description conceptuelle d'une base de données en projet.

En général, cette intégration est réalisée pendant l'analyse conceptuelle; dans ce cas, son but est de produire un schéma intégré à partir de plusieurs schémas d'applications qui ont été produits indépendemment.

I.2.2.2 Approche "intégration de bases de données existantes"

Elle produit le schéma global d'un ensemble de bases de données.
On peut distinguer :
a) les bases de données homogènes, c'est à dire travaillant sur le même modèle de données et le même SGBD.

b) les bases de données hétérogènes, c'est à dire travaillant sur des modèles de données différents. Cette hypothèse complique le travail car il est nécessaire de passer par un modèle conceptuel commun comme dans le système **SCOOP** [Spaccapietra, Demo, Parent, 83].

I.2.3 Les résultats obtenus.

En plus du schéma intégré, certaines méthodes fournissent les mappings entre ce schéma et les vues des utilisateurs [Belfar], [Biskup], [Lenzerini].
Le tableau 3 compare les différentes méthodes sur ce point.

I.2.4 Le degré d'interactivité

Ce point dépend principalement de l'optique de travail choisie ainsi que de l'existence d'une aide à la détection des correspondances inter-schémas.

I.2.5 Les stratégies d'intégration

Nous distinguons deux familles de stratégies:

a) **intégration binaire** qui intègre n schémas deux par deux, elle se subdivise en deux sous types

-l'**intégration en échelle** qui travaille sur un schéma intermédiaire auquel on intègre à chaque pas un nouveau composant.

-le **mode équilibré** qui intègre les schémas par paires et de façon symétrique.

b) **intégration n-aire** qui intègre en un seul pas n schémas (n>2).

La figure 2.1 schématise les différentes stratégies.

Bien sur, les stratégies binaires simplifient la recherche des correspondances à chaque pas de l'intégration (on peut montrer que la complexité de l'algorithme de fusion pour n schémas est d'ordre n^2); seulement, le nombre d'opérations d'intégration s'accroît en conséquence.
D'autre part, l'intégration n-aire permet de commencer par traiter les correspondances les plus

simples, ce qui est préférable car les correspondances compliquées ne sont parfois que la composition de correspondances simples.

Méthode / Concept	Belfar	Navathe/ElMasri	Batini/Lenzerini
Entité	OUI	OUI	OUI
Association	OUI	OUI	OUI
Attribut	Complexe ou simple optionnel ou obligatoire mono ou multivalué	Simple,monovalué	Simple Mono ou multivalués
Généralisation	NON	Catégorie(entité) Inclusion(association)	Sous-type Sous-ensemble
Sous-type	NON	NON	Sous-type Sous-ensemble
Cardinalité	Oui mais uniquement pour les entités dans une relation	Oui mais uniquement pour les entités dans une relation	OUI
Contraintes d'intégrité	Celles impliquées par les cardinalités	Celles impliquées par les cardinalités	Celles impliquées par les cardinalités

Concepts des différents modèles

Tableau 1

Correspondances sémantiques	Belfar	Navathe/ ElMasri	Batini/ Lenzerini
Structures et populations identiques, éventuellement les noms des structures différents.	OUI	OUI	OUI
Structures de même type mais avec des populations différentes et des caractéristiques spécifiques à chacune	NON	OUI (Spécialisation)	OUI (sous-type, sous-ensemble)
Structures différentes mais à sémantiques équivalentes	OUI	OUI	OUI
Structures décrivant les mêmes objets du monde réel sous des angles de vue différents (ensembles d'attriburts différents)	OUI	OUI	OUI
Structures décrivant les mêmes objets du monde réel à des degrés de précision différents	OUI	OUI	OUI
Structures décrivant les mêmes types d'objets mais avec des contraintes d'intégrité différentes	OUI	NON	NON

Correspondances sémantiques prises en compte dans les différentes approches

Tableau 2

OBJECTIFS	Belfar	Navathe/ ElMasri	Batini/ Lenzerini
Obtention d'un schéma conceptuel	OUI	OUI	OUI
Mapping schéma conceptuel/vues	OUI	NON	NON
Recherche des correspondances	NON	OUI	OUI

Objectifs des différentes méthodes

Tableau 3

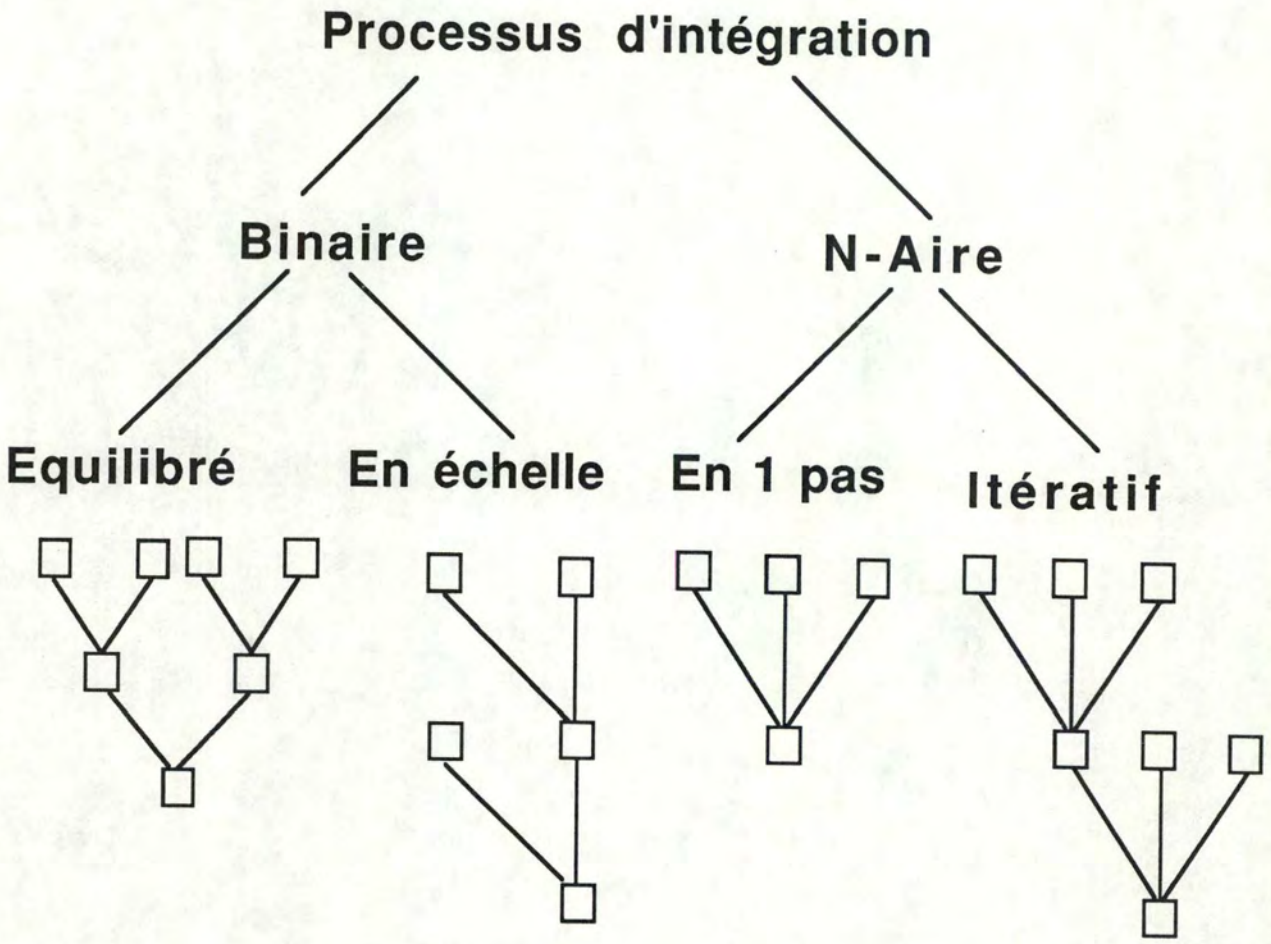


Figure 2.1: **Stratégies d'intégration**

Chapitre I.3: TYPOLOGIE GENERALE DES CORRESPONDANCRES INTER-SCHEMAS

I.3.1 Identité

I.3.2 Equivalence

I.3.3 Compatibilité

I.3.4 Sous-types

I.3. TYPOLOGIE GENERALE DES CORRESPONDANCES INTER-SCHEMAS

Nous allons, dans ce chapitre, développer une typologie de correspondances inter-schémas ne dépendant d'aucun modèle conceptuel particulier.

I.3.1 Identité

=> Les différents concepteurs modélisent un même ensemble de propriétés d'une même classe d'objets du monde réel par des structures identiques.

Eventuellement l'intitulé des objets peut être différent, nous parlerons alors de synonymes.
Il est à noter qu'une gestion des synonymes peut représenter une aide efficace dans la détection de correspondances.

I.3.2 Equivalence

=> Les différents concepteurs modélisent un même ensemble de propriétés d'une même classe d'objets du monde réel par des structures différentes.

On peut donner comme raison à cela les objectifs généralement différents de plusieurs concepteurs. Chacun modélisant pour satisfaire les buts du sytème au sein duquel il travaille.

I.3.3 Compatibilité

=> Les différents concepteurs modélisent un ensemble différent de propriétés d'une même classe d'objets du monde réel par des structures différentes.
Nous pouvons subdiviser cette correspondance en :

I.3.3.1 angle de vue

=> Les différents ensembles de propriétés ne sont ni identiques, ni inclus l'un dans l'autre.

I.3.3.2 Degré de précision

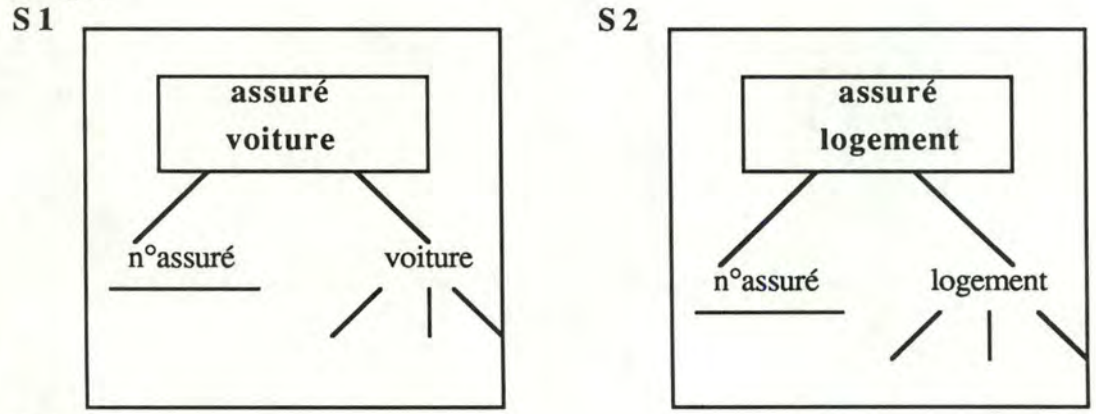
=> Une structure est plus précise qu'une autre si on peut déduire la population de la seconde de la celle de la première.[Grison 87]

Un ensemble de propriétés, modélisé par la structure la moins précise, est inclus dans un autre modélisé par une structure plus précise.

I.3.4 sous-types

=> Les concepteurs modélisent un ensemble de propriétés éventuellement différent appartenant à des classes d'objets du monde réel dont certaines (sous-types) sont incluses dans une autre (type).

exemple:



[Grison 87]

Certains assurés peuvent faire partie des deux populations, nous nous trouverions donc devant le type assuré dont assuré logement et assuré voiture sont les sous-types.

Chapitre 4: TYPOLOGIE RELATIVE AU MODELE ERA

I.4.1 Attribut/Attribut

I.4.2 Entité/Entité

I.4.3 Relation/Relation

I.4.TYPOLOGIE RELATIVE AU MODELE ERA

Pour chaque type de correspondance nous allons donner une liste de critères qui permettront de l'identifier. Notre intention est que cette typologie recouvre un maximum de correspondances sémantiques, cependant, vouloir les détecter absolument toutes risque de donner lieu à des critères trop complexes. Il est donc permis d'oublier certains cas rares de correspondances, ce qui entraine de la redondance dans le schéma intégré, le principal est de détecter les cas les plus fréquents.

I.4.1 Attribut/Attribut : AA

I.4.1.1 Types de correspondances **AA**

a] ***identité***

ET

- Les deux attributs appartiennent à des objets (Entité ou Relation) en correspondance.
- Ils modélisent une même propriété de ces objets.
- **OU exclusif**
  - **ET**
    - Ils sont tous deux non décomposables
      - Ils ont le même domaine de définition.
      - Leurs cardinalités sont identiques.
  - **ET**
    - Ils sont tous deux décomposables.
    - Le nombre d'attributs composants est le même pour les deux attributs
    - La correspondance "identité" forme une bijection sur les ensembles des attributs composants.
    - Leurs cardinalités sont identiques

Correspondance: identique.

b] ***précision***

ET

- Les deux attributs appartiennent à des objets (Entité ou Relation) en correspondance.
- Ils modélisent une même propriété de ces objets.
- **OU exclusif**
  - **ET**
    - Ils sont tous deux non décomposables
    - Leurs types sont soit identiques soit dérivables l'un de l'autre.
    - **OU inclusif**
      - Le type de l'attribut le moins précis est dérivable de celui du plus précis (ex: arrondi 14,53 et 14, entier inclus dans réel).
      - L'intervalle représentant la cardinalité du plus précis est inclus dans celui représentant la cardinalité du moins précis.

  - **ET**
    - Ils sont tous deux décomposables
    - **ET**
      - **OU exclusif**
        - **et**

        -Le nombre de composants de l'attribut le moins précis est < au nombre de composants de l'attribut le plus précis.

        -Tout attribut composant de l'attribut le moins précis est en correspondance "identité" ou "moins précis que" avec un attribut composant de l'attribut le plus précis.
        - **et**

        -Le nombre de composants de l'attribut le moins précis est = au nombre de composants de l'attribut le plus précis.

        -Au moins un des attributs composants est "moins précis que" son correspondant de l'attribut le plus précis et aucun n'est le point d'arrivée d'une correspondance "moins précis que".
        - L'intervalle représentant la cardinalité du plus précis (déterminé par le OU exclusif ci-dessus) est inclus dans ou égal à celui du moins précis.

Correspondance :moins précis que

c]_*angle de vue*

OU exclusif

- ET

- Les attributs en correspondance sont non décomposables
- Les différences des intervalles représentant les cardinalités sont non vides.
- Ils appartiennent à des objets en correspondance.
- Ils modélisent une même propriété de ces objets.

- ET

- Les attributs en correspondance sont tous deux décomposables .
- Ils appartiennent à des objets en correspondance.
- Ils modélisent une même propriété de ces objets.
- Soient A1 l'ensemble des attributs composants du premier attribut et A2 celui du deuxième.

OU exclusif

- Il existe dans les deux attributs des composants non en correspondance.

- et

- Si l'ensemble des correspondances AA forme une application de A1 vers A2.
  - qu'il existe un composant de A2 qui n'est pas en correspondance avec un composant de A1.
  - qu'il existe au moins un composant de A2 qui est en correspondance "moins précis que" avec un composant de A1.

- et

- Si l'ensemble des correspondances AA forme une application de A2 vers A1.
- qu'il existe un attribut de A1 qui n'est pas en correspondance avec un attribut de A2.
- qu'il existe au moins un attribut de A1 qui est en correspondance "moins précis que" avec un attribut de A2.

- et

- Si l'ensemble des correspondances AA forme une bijection entre A2 et A1
- qu'il existe au moins un attribut de A1 qui est en correspondance "moins précis que" avec un attribut de A2.
- qu'il existe au moins un attribut de A2 qui est en correspondance "moins précis que" avec un attribut de A1.

- **et**
  - Il existe une correspondance "angle de vue" entre deux attributs composants des attributs complexes en correspondance.

- **et**
  - Si toutes les conditions de l'identité entre attributs décomposables sont réunies à l'exception de celle portant sur les cardinalités.
  - Ces dernières sont telles que les différences des intervalles les représentant sont non vides.

- **et**
  - Si toutes les conditions de la précision entre attributs décomposables sont réunies à l'exception de celle portant sur les cardinalités.
  - Ces dernières sont telles que l'intervalle représentant la cardinalité du plus précis (déterminé par la condition ci-dessus) inclus celui représentant la cardinalité du moins précis.

Correspondance : angle différent

REMARQUE:

Nous n'avons pas présenté le cas d'une correspondance entre un attribut complexe et un attribut simple. On peut admettre l'existence d'un lien sémantique entre ces deux types d'attributs. Cependant, cela implique que l'attribut simple est en correspondance avec un des composants de l'attribut complexe. Nous pensons que décrire des correspondances complexe/simple n'apporterait pas d'informations utiles à l'intégration mais bien un surcroît de difficulté. Nous revenons donc à une des correspondances décrites précédemment.

d] ***fonction***

ET

- L'attribut déterminé et l'ensemble des attributs déterminants appartiennent respectivement à deux objets en correspondance.
- L'attribut déterminé et l'ensemble des attributs déterminants modélisent une même propriété des objets en correspondance.
- L'attribut déterminé est obtenu par une fonction donnée sur l'ensemble des attributs déterminants.
- La fonction inverse doit exister et être donnée.
- Les attributs sont non décomposables.

Remarque:
déterminé = Output de la fonction
déterminant = Input de la fonction

exemple : S1 attr1 = **taille en mètres** S2 attr2 = **taille en pieds**

$f(attr1) = attr1*3 = attr2$
$f^{-1}(attr2) = attr2/3 = attr1$

Un cas particulier intéressant de la fonction est celui où un attribut d'un objet est l'**agrégat** d'un ensemble d'attributs d'un objet en correspondance.
Agréger = "Réunir en un tout des parties sans liaisons naturelles"[Petit Larousse]
Dans ce cas, le produit cartésien des domaines des attributs composants est égal au domaine de l'attribut agrégat; et la fonction est une concaténation.

exemple : S1 attr1 = **adresse** S2 attr2 = **rue**
attr3 = **numéro**
attr4 = **localité**

On considère l'adresse comme un agrégat représentant la rue, le numéro et la localité.
Si, par contre, l'adresse était l'agrégat de rue, numéro, localité ET pays, nous ne nous trouverions plus devant une fonction d'agrégation.

Représentons la situation par des relations entre ensembles:

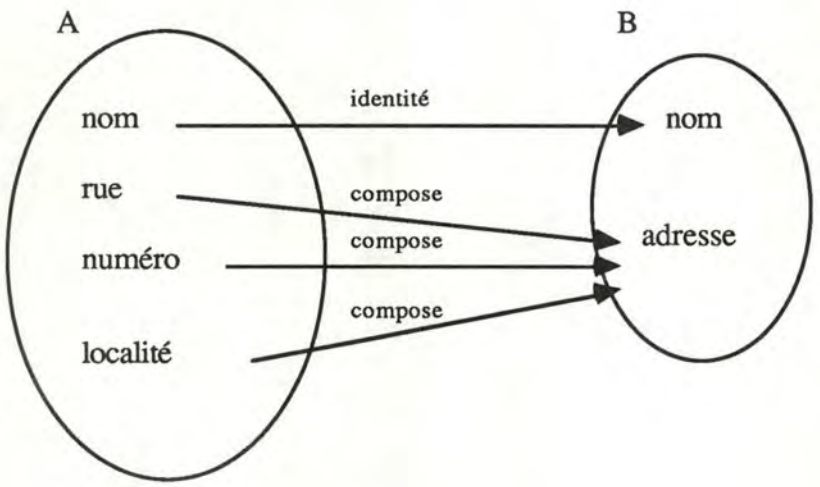


schéma 1

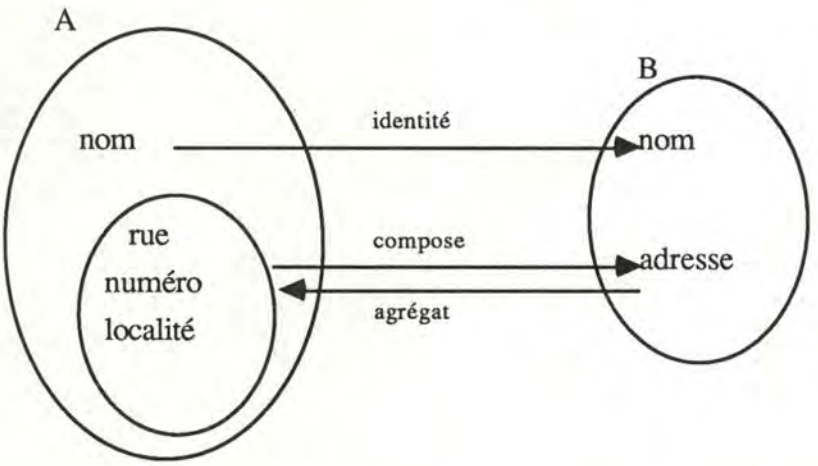


schéma 2

Pour plus de facilité, nous choisirons la représentation du schéma 2. Ce qui nous semble logique car dans le schéma 1, chaque correspondance est indépendante des autres et on ne peut pas retrouver l'agrégat à partir d'un seul de ses composants
De ce fait, le sous-ensemble composant l'attribut agrégat **se comporte comme un et un seul attribut**.

Il est important de disposer à la fois de la fonction et de son inverse car si la fonction inverse n'existait pas, il serait impossible de réaliser l'intégration.
En effet, dans le cas adresse, par exemple, si on choisit d'intégrer en conservant l'agrégat, il sera impossible d'obtenir par la suite les valeurs individuelles. Si on intègre en conservant les valeurs individuelles, le problème se posera lors de l'introduction d'une occurrence sous forme d'agrégat.

De manière à ne pas limiter à priori la complexité des fonctions, nous demanderons qu'elles soient **données sous forme de procédures**. Cette contrainte supplémentaire vis-à-vis de l'utilisateur est la seule manière de résoudre certains cas tels que l'agrégation.

I.4.1.2 Propriétés des correspondances AA

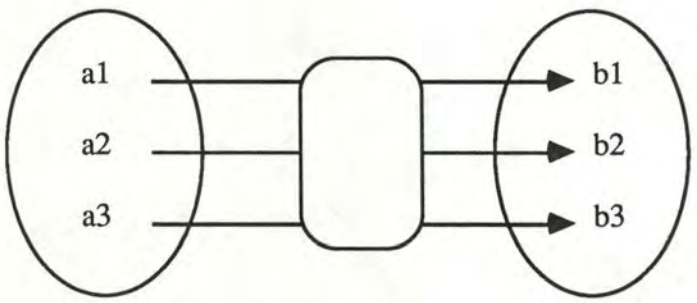
Dans ce qui va suivre, nous allons utiliser les notations ensemblistes suivantes:
Soient deux objets E1 et E2, nous allons grouper leurs attributs **non décomposables** dans deux ensembles; respectivement A et B.
Entre ces ensembles, nous établirons des liaisons, notées f(), qui représenteront les correspondances au sens global du terme. Seul nous intéresse le fait de connaître les attributs qui ne sont pas visés par une correspondance. Les combinaisons possibles des correspondances n'ayant

pour l'instant aucun intérêt, nous l'illustrerons par les boites rassemblant les flêches joignant les deux ensembles.
=> Les correspondances globales, appelées correspondances simples dans la suite par oposition aux correspondances précises décrites jusqu'à présent, sont des correspondances dont on ne précise pas le type (identité, précision, fonction).

Voyons maintenant les principales figures que peuvent donner les liaisons entre A et B.

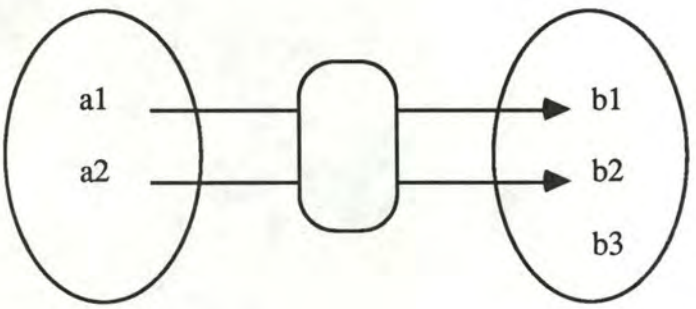
a] ***propriété A***

Pour tout a appartenant à A : il existe f(a) qui appartient à B
et Pour tout b appartenant à B: il existe $f^{-1}(b)$ qui appartient à A



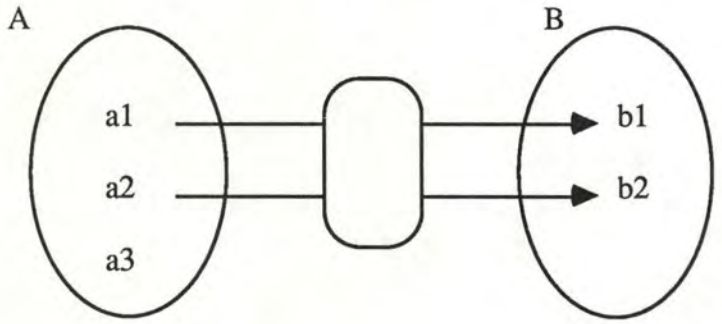
b] ***propriété B***

Pour tout a appartenant à A : il existe f(a) qui appartient à B
et Il existe b qui appartient à B : $f^{-1}(b)$ n'existe pas



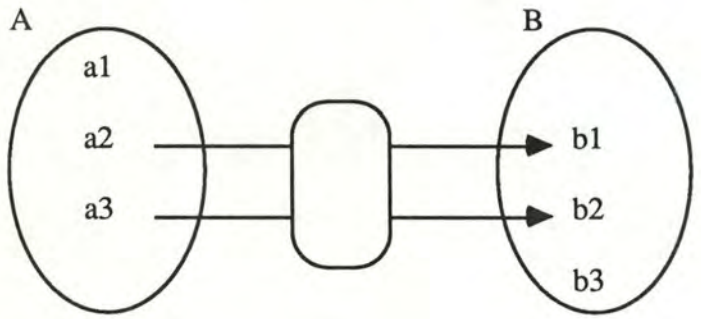
c] ***propriété C***

Pour tout b appartenant à B: il existe $f^{-1}(b)$ qui appartient à A
et Il existe a qui appartient à A : f(a) n'existe pas



d] ***propriété D***

Il existe a qui appartient à A : f(a) n'existe pas
et Il existe b qui appartient à B : $f^{-1}(b)$ n'existe pas



I.4.1.3 priorités entre correspondances

Définissons un ordre de priorité entre correspondances AA.
Par ordre croissant:

identité ou fonction
plus ou moins précis que
angle de vue

Nous pourrons dés lors, à partir de l'ensemble des correspondances entre attributs non décomposables de deux objets, obtenir une correspondance résultante qui nous sera utile dans la suite.

La correspondance "fonction" est sur le même niveau que l'identité car la fonction devant exister dans les deux sens, le contenu sémantique des déterminés et déterminants est le même. La fonction est donc une "identité" sans structures identiques, c'est à dire une équivalence.

La correspondance résultante sera celle de priorité la plus élevée avec un cas particulier lorsqu'on trouve comme correspondances de degré le plus élevé,"moins précis que" dans les deux sens, la résultante sera alors angle de vue.

I.4.1.4 Utilité des propriétés et priorités.

On peut envisager deux types d'utilisation de ces propriétés:

a) *Aide à la détection des correspondances*

On peut s'en servir pour guider l'utilisateur dans l'examen des points importants dont dépend le type d'une correspondance. C'est un outil d'aide et de contrôle et non un outil entièrement automatisable car ici encore, la place de la sémantique est trop importante dans la nature des correspondances entre concepts.
Un exemple sera donné pour les correspondances EE.

b) *Validation d'un ensemble de correspondances entre schémas*

Si on se place dans l'optique d'un ensemble de correspondances donné par l'utilisateur, on peut effectuer un contrôle de cohérence sur ces correspondances. L'application de cet outil sur les correspondances entre attributs simples doit donner les mêmes correspondances EE que celles introduites par l'utilisateur.

I.4.2 Entité/Entité : EE

I.4.2.1 Liste des correspondances EE

REMARQUES IMPORTANTES:

- Lorsque le nombre des attributs intervient dans une définition, il faut se souvenir que nous considérons les attributs déterminants ou déterminés d'une fonction comme un seul élément.
- La correspondance AA-fonction étant de même niveau que AA-identité, chaque fois que nous utiliserons l'expression "correspondance identité entre attributs", il faudra la généraliser aux correspondances fonctions.

a] ***identité***

ET

- Leur population modélise les mêmes objets du monde réel.
- Le nombre d'attributs non décomposables est le même pour les deux entités
- La correspondance "identité" forme une bijection sur les ensembles des attributs des entités en correspondance.

b] ***degré de précision***

ET

- Leur population modélise les mêmes objets du monde réel.
- **OU exclusif**
  - **et**
    - Le nombre d'attributs non décomposablesde l'entité la moins précise est < au nombre d'attributs non décomposables de l'entité la plus précise.
    - Tout attribut de l'entité la moins précise est en correspondance "identité" ou "moins précis que" avec un attribut de l'entité la plus précise.
  - **et**
    - Le nombre d'attributs non décomposables de l'entité la moins précise est = au nombre d'attributs non décomposables de l'entité la plus précise.
    - Au moins un des attributs est "moins précis que" son correspondant de l'entité la plus précise et aucun n'est le point d'arrivée d'une correspondance "moins précis que".

Correspondance : moins précis que

c] ***type /sous-type***

Un type d'entité d'un des schémas est un sous-type d'un type d'entité du second schéma.

- L'ensemble des objets modélisés par la population du sous-type est inclus dans celui modélisé par la population du type.

Correspondance : sous-type de

d] ***sous-type/sous-type***

Certains types d'entité peuvent être perçus comme sous-type d'un type d'entité plus général n'étant repris dans aucun des schémas à intégrer.

ET

- L'intersection des ensembles d'objets modélisés par les populations des deux sous-types n'est pas égale à leur union.
- L'ensemble des objets modélisé par le type général comprend les ensembles modélisés par les sous-types.

exemple: S1 **VW** S2 **Audi**

Le type voiture n'existant dans aucuns des schémas à intégrer.

Correspondance : sous-type/sous-type

e] ***angle de vue***

ET

- Leurs populations modélisent les mêmes classes d'objets.
- Soient A1 l'ensemble des attributs de la première entité et A2 celui de la deuxième.

OU exclusif

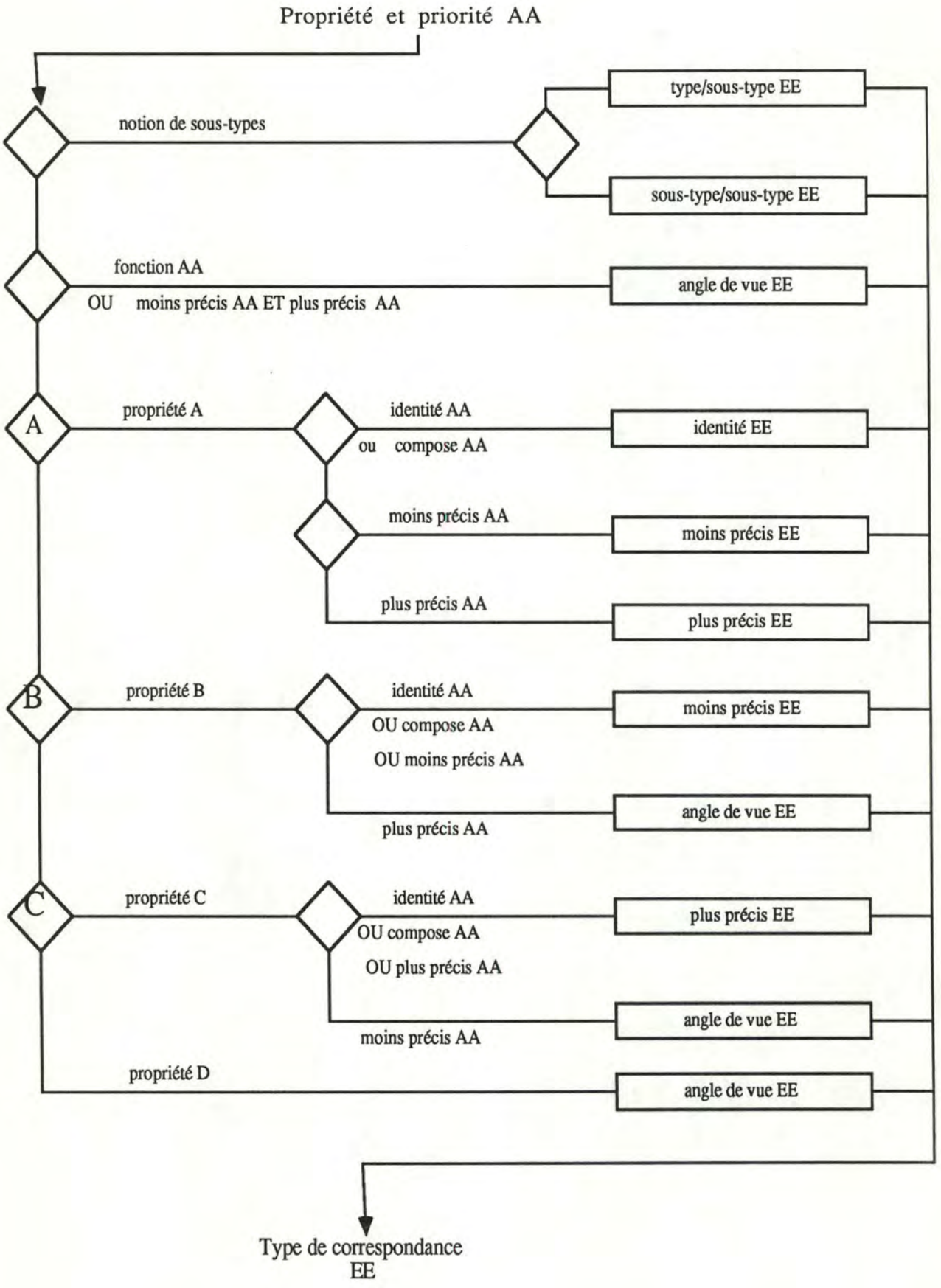
- Il existe dans les deux entités des attributs non en correspondance.
- **et**
- Si l'ensemble des correspondances AA forme une application de A1 vers A2.
  - qu'il existe un attribut de A2 qui n'est pas en correspondance avec un attribut de A1.
  - qu'il existe au moins un attribut de A2 qui est en correspondance "moins précis que" avec un attribut de A1.

- **et**
  - Si l'ensemble des correspondances AA forme une application de A2 vers A1.
  - qu'il existe un attribut de A1 qui n'est pas en correspondance avec un attribut de A2.
  - qu'il existe au moins un attribut de A1 qui est en correspondance "moins précis que" avec un attribut de A2.
- **et**
  - Si l'ensemble des correspondances AA forme une bijection entre A2 et A1
  - qu'il existe au moins un attribut de A1 qui est en correspondance "moins précis que" avec un attribut de A2.
  - qu'il existe au moins un attribut de A2 qui est en correspondance "moins précis que" avec un attribut de A1.
- **et**
- Il existe au moins deux attributs en correspondance "angle de vue".

Correspondance : angle de vue

I.4.2.2 détection du type de correspondance EE

Voyons un exemple d'utilisation des propriétés des correspondances AA dans la détection des correspondances EE.
Si l'on excepte les cas de sous-types qui font trop appel à la sémantique, il est possible de détecter le type de la correspondance EE à partir des correspondances entre les attributs des entités en correspondances.
Pour ce faire l'algorithme de détection décrit ci-dessous, qui est déduit de la liste des propriétés des correspondances EE, fait appel à deux données: d'une part le type de figure formée par l'ensemble des correspondances AA au sens global du terme (A,B,C ou D) et d'autre part la résultante des correspondances AA. Ces données ont été décrites précédemment.


Propriété et priorité AA
notion de sous-types
type/sous-type EE
sous-type/sous-type EE
fonction AA
OU moins précis AA ET plus précis AA
angle de vue EE
A
propriété A
identité AA
ou compose AA
identité EE
moins précis AA
moins précis EE
plus précis AA
plus précis EE
B
propriété B
identité AA
OU compose AA
OU moins précis AA
moins précis EE
plus précis AA
angle de vue EE
C
propriété C
identité AA
OU compose AA
OU plus précis AA
plus précis EE
moins précis AA
angle de vue EE
propriété D
angle de vue EE
Type de correspondance
EE

I.4.3 relation/relation : RR

Pour que deux relations soient en correspondance, il faut qu'au moins deux de leurs rôles aient une même sémantique, c'est à dire qu'ils relient deux entités en correspondance et que leurs dénominations soient synonymes. L'égalité de leurs cardinalité n'est pas nécessaire.

a] ***identité***

Soient E1, l'ensemble des n entités e1j, j = 1...n reliées par la relation R1;
E2 m e2k, k = 1...m R2;
r1j, le rôle de e1j dans R1
r2k e2k R2
on a:

ET
- Les populations de R1 et R2 doivent modéliser les mêmes interactions entre objets du monde réel
- L'ensemble des correspondances entre E1 et E2 doit former une bijection
- pour tout e1j, e2k en correspondance, leurs populations doivent modéliser les mêmes objets du réel
- r1j doit être synonyme de r2k.
- card(r1j) = card(r2k).
- La correspondance "identité" forme une bijection sur les ensembles des attributs des relations en correspondance.

b] ***précision***

ET

- Les populations de R1 et R2 doivent modéliser les mêmes interactions entre objets du monde réel
- Toute entité de la relation la moins précise est en correspondance avec une entité différente de la relation la plus précise.
- Les rôles des entités en correspondance sont synonymes.
- Les populations de deux entités en correspondance modélisent les mêmes objets du monde réel.

- **OU inclusif**
  - Le degré de la relation la moins précise est <= à celui de la plus précise.
  - Pour au moins un des rôles des entités de la relation la moins précise l'intervalle exprimant sa cardinalité inclu celui du rôle correspondant dans la relation la plus précise et pour aucun rôle des entités de la relation la plus précise l'intervalle exprimant sa cardinalité n'est inclu celui du rôle correspondant dans la relation la moins précise.
  - Le nombre d'attributs non-décomposables de la relation la moins précise est < à celui dela plus précise.
  - Le nombre d'attributs non-décomposables de la relation la moins précise est = à celui de la plus précise et il existe pour au moins un des attributs une correspondance "moins précis que" avec l'un des attributs de la relation la plus précise de plus aucun d'entre eux n'est le point de départ d'une correspondance "plus précis que".

Correspondance : plus précis que

c] ***angle de vue***

ET

- Les populations de R1 et R2 doivent modéliser les mêmes interactions entre objets du monde réel
- Les rôles des entités en correspondance sont synonymes.
- **OU inclusif**
  - Il existe dans les deux relations au moins une entité qui n'est pas en correspondance avec une de l'autre relation.
  - Au moins une des cardinalités de la première relation est plusprécise que sa correspondante dans la seconde relation et au moins une des cardinalités de la seconde relation est plus précise que sa correspondante dans la première.
  - Soient A1 l'ensemble des attributs de la première relation et A2 celui de la deuxième.

    OU exclusif
    - Il existe dans les deux relations des attributs non en correspondance.
    - **et**
    - Si l'ensemble des correspondances AA forme une application injective de A1 vers A2.
      - qu'il existe un attribut de A2 qui n'est pas en correspondance avec un attribut de A1.
      - qu'il existe au moins un attribut de A2 qui est en correspondance "moins précis que" avec un attribut de A1.

- **et**
- L'ensemble des correspondances AA forme une application injective de A2 vers A1.
  - qu'il existe un attribut de A1 qui n'est pas en correspondance avec un attribut de A2.
  - qu'il existe au moins un attribut de A1 qui est en correspondance "moins précis que" avec un attribut de A2.
- **et**
  - L'ensemble des correspondances AA forme une bijection entre A2 et A1
  - qu'il existe au moins un attribut de A1 qui est en correspondance "moins précis que" avec un attribut de A2.
  - qu'il existe au moins un attribut de A2 qui est en correspondance "moins précis que" avec un attribut de A1.

Correspondance : angle différent

d] ***type/sous-type***

ET

- Tous les types d'entité impliqués dans la relation la plus générale sont en correspondance avec un type d'entité différent de la relation la plus spécialisée
- L'ensemble d'interactions entre objets du monde réel modélisé par la population de R1 doit contenir celui modélisé par la population de R2

Cela est vrai notamment lorsque parmis les correspondances entre les entités impliquées dans les relations, on trouve une ou plusieurs correspondances type/sous-type dans le même sens.

e] ***sous-type/sous-type***

Plusieurs relations sont en correspondance sous-type/sous-type si on peut les mettre en correspondance type-sous/type avec un même type de relation.

I.4.4 Correspondances mixtes

ET

- Les ensembles d'objets modélisés par les deux populations sont identiques.
- En transformant la structure la moins générale en structure de même type que la plus générale, on peut établir entre ces deux structures une correspondance identité, précision ou angle de vue. Les notions de sous-types étant éliminées par le premier point.

Correspondance : EA, RA, ER

Chapitre I.5: MAPPING STRUCTUREL

I.5.1 Utilisation lors de l'intégration

I.5.2 Exemples

I.5. MAPPING STRUCTUREL

=> mapping structurel : c'est l'ensemble des correspondances entre deux schémas.

I.5.1 Utilisation lors de l'intégration

Un mapping est défini entre chaque paire de schémas à intégrer.
Soient les schémas S1 , S2 et S3.
Nous disposons donc des mappings structurels, soient **MS(S1,S2),MS(S1,S3),MS(S2,S3)**.
En vue de constituer les mappings opérationnels nous devons construire les mappings suivants : MS(SC,S1) , MS(SC,S2) et MS(SC,S3), avec SC désignant le schéma résultant de l'intégration.

I.5.2 Syntaxe de désignation des correspondances (S.D.C.)

Le formalisme de notre syntaxe sera basé sur celui développé par Mme K. **Belfar** [Belfar 84].

E : désignera une entité
A : " " un attribut
R : " " une relation

Les correspondances seront désignées :

EE : désignera une correspondance Entité/Entité
RR : " " " Relation/Relation
AA : " " " Attribut/Attribut
EA : " " " Entité/Attribut
RA : " " " Relation/Attribut
ER : " " " Entité/Relation
ES : " " " Entité/Structure
RS : " " " Relation/Structure

MS(S1,S2) : désignera le mapping structurel entre S1 et S2, il est composé des correspondances structurelles dans les deux sens.
CS(S1,S2) : désignera une correspondance structurelle **de** S1 **vers** S2

[] signifie : facultatif
{} signifie : répétitif
| signifie : OU

mapping::= MS(<nom de schéma 1>,<nom de schéma 2>) ;
CS(<nom de schéma 1>,<nom de schéma 2>) ; <liste des correspondances>
CS(<nom de schéma 2>,<nom de schéma 1>) ; <liste des correspondances>

Remarque: CS(schéma origine, schéma destination)

<liste des correpondances> ::= {<correspondance> ; }

<correspondance>::=<type EE> | <type AA> | <type RR> | <type ER >|<type EA> |<type RA>|

<type EE>::=EE <libellé de l'entité du schéma origine>*<type de correspondance EE>*<libellé de l'entité du schéma destination>

<type AA>::=AA <libellé de l'attribut du schéma origine>.<libellé de l'objet possédant l'attribut>*<type de correspondance AA>*<libellé de l'attribut du schéma destination>.<libellé de l'objet possédant l'attribut>

<type RR>::=RR <libellé de la relation du schéma origine>*<type de correspondance RR>*<libellé de la relation du schéma destination>

<type ER >::=ER <libellé de l'entité du schéma origine> * <libellé de la relation du schéma destination>

<type EA>::=EA <libellé de l'entité du schéma origine>*<libellé de l'attribut> . <libellé de l'objet du schéma destination possédant l'attribut>

<type RA>::=RA <libellé de la relation du schéma origine> * <libellé de l'attribut> . <libellé de l'objet du schéma destination possédant l'attribut>

<type de correspondance EE>::=identité | sous-type de | sous-type/sous-type | moins précis que | angle de vue | globale

<type de correspondance RR>::=<type de correspondance EE>

<type de correspondance AA>::=identité fonction <définition de la fonction> | moins précis que | angle de vue | globale

<définition de la fonction>::= Procédure

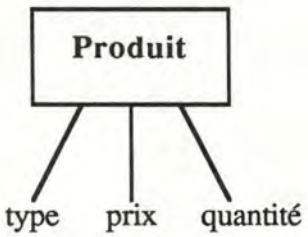
<libellé d'objet>::=<libellé d'un attribut> . <libellé de l'objet possédant cet attribut> | <libellé d'entité> | <libellé de relation>

Dans notre syntaxe, le nom des attributs est unique au sein des entités qui les possèdent mais non dans leur schéma.
Les noms des entités et relations sont uniques au sein de leur schéma.

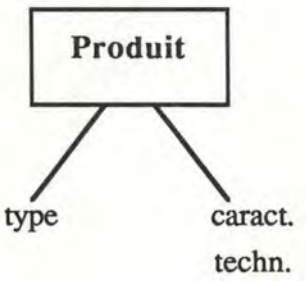
5.3 Exemples de mappings structurels

5.3.1 Soient les schémas S1 et S2,

S1: stocks



S2: production

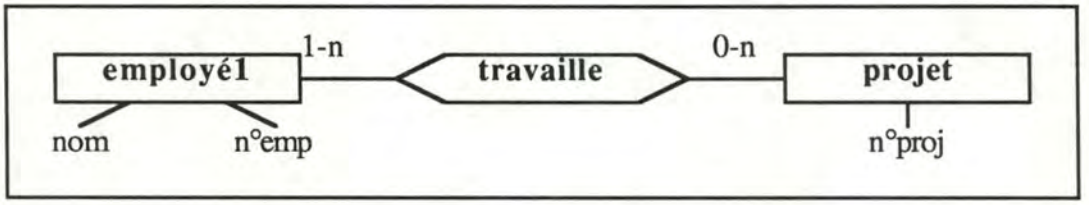


Les entités produit représentent un même concept, mais vu de deux manières différentes. Disons que S1 est l'optique d'un département de gestion des stocks, tandis que S2 est celui de la production.
Nous nous trouvons donc devant une correspondance Entité/Entité avec angles de vue différents.
Sous la forme de la syntaxe :

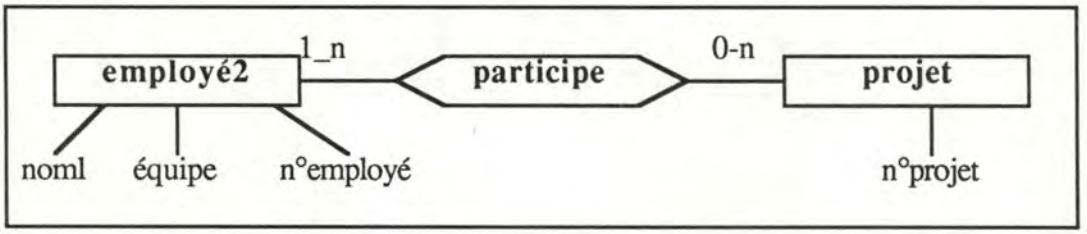
MS(S1,S2);
CS(S1,S2);
EE produit*angle de vue*produit;
AA type.produit*identité*type.produit;
CS(S2,S1);
EE produit*angle de vue*produit;
AA type.produit*identité*type.produit;

5.3.2 Soient les schémas S1 et S2 :

S1



S2



Pour qu'il y ait correspondance entre deux relations il faut une correspondance entre au moins deux des entités qu'elles relient.
C'est le cas ici, l'entité employé1 correspond à l'entité employé2 sous un angle de vue différent, de plus il y a identé entre les deux entités projet.
Pour terminer, supposons une correspondance entre la relation travaille de S1 et participe de S2.

Dans la syntaxe :
MS(S1,S2);
CS(S1,S2);
EE projet*identité*projet;
EE employé1 * moins précis que *employé2;
RR travaille*identité*participe;
AA n°proj.projet*identité*n°projet.projet;
AA nom.employé1*identité*nom.employé2;

AA n°emp.employé1*identité*n°employé.employé2;
CS(S2,S1);
EE projet*identité*projet;
EE employé2 * moins précis que *employé1;
RR participe *identité* travaille;
AA n°projet.projet*identité*n°proj.projet;
AA nom.employé2*identité*nom.employé1;
AA n°employé.employé2*identité*n°emp.employé1;

Chapitre I.6: COMPARAISON INTEGRATION DE VUES / INTEGRATION DE BD

I.6.1 La résolution des conflits

I.6.2 Recherche des correspondances

I.6.3 Mapping opérationnel

I.6.4 Conclusion

I.6. COMPARAISON: INTEGRATION DE SCHEMAS / INTEGRATION DE BD

I.6.1 La résolution des conflits.

Dans l'intégration de bases de données existantes, on peut difficilement se permettre de modifier les différents schémas externes car il faudrait aussi modifier leurs extensions; d'autre part, on peut supposer qu'il existe déjà des programmes d'application. De telles modifications représenteraient des coûts trop élevés. Cela revient à supposer que tous les schémas externes sont des modélisations correctes de la réalité. Ayant déja été utilisés nous pouvons dire qu'ils ont fait leurs preuves du moins dans l'optique de leur concepteur.

Il résulte de cette hypothèse que la résolution des conflits se fait plutôt en choisissant la structure la plus générale dans le schéma intégré, ou en utilisant le concept de généralisation.

Dans l'intégration de schémas conceptuels, au contraire, le but est plutôt de trouver la représentation la plus fidèle possible de la réalité grace à une décomposition du travail de modélisation. Dans cette optique, rien n'empêche qu'un utilisateur modifie sa vue du problème pour obtenir une meilleure sémantique.
La résolution des conflits pourra donc se faire par modification des schémas conceptuels.
Evidemment, il est aussi possible que deux structures différentes modélisent valablement un même concept

Quatre cas sont à envisager :

6.1.1 Identité

Dans ce cas, il sagit d'une simple fusion, c'est à dire que les concepts identiques représentent en fait un seul et même concept qui est placé dans le schéma intégré.

6.1.2 Equivalence

Dans les deux approches, la solution est d'adopter dans le schéma global la structure la plus générale.

6.1.3 Compatibilité

Dans l'approche "intégration de bd existantes", la solution est d'adopter dans le schéma global la structure modélisant toutes les informations reprises dans les différentes structures à intégrer de façon à réaliser l'objectif de complétude.

Dans l'autre approche une autre possibilité est offerte, c'est la modification de schéma(s). Soit un des concepteurs a manqué de précision, par exemple en cas de cardinalité différente, soit il a commis une erreur.

Par un dialogue il faut amener le ou les concepteurs fautifs à améliorer leurs schémas de manière à se retrouver dans un des deux cas précédents.

6.1.4 Sous-types

Dans les deux approches, il faut créer dans le schéma intégré les structures modélisant correctement cette correspondance.

I.6.2 Recherche des correspondances

Nous n'avons noté aucune différence sur ce point entre les deux approches.

I.6.3 Mappings opérationnels

Deux problèmes sont à étudier concernant l'accès aux occurrences:

6.3.1 Violation d'une contrainte

Illustrons par un exemple : la cardinalité d'une entité E1 impliquée dans la relation R est 1-1 dans le schéma S1 et 0-1 dans le schéma S2 qui contient une entité et une relation identiques à celles de S1.

Ce qui signifie qu'un programme d'application prévu pour tourner sur le schéma S1 attend toujours un occurence de R.

Si nous avons intégré des BD existantes un problème va se poser, en effet, nous prendrons l'expression la plus générale de la cardinalité c'est à dire 0-1 donc on pourra trouver des entités E1 n'apparaissant pas dans une relation R. Ce qui peut signifier l'arrêt de programmes prévus pour S1 à moins que l'on ai prévu, lors de l'élaboration des mappings opérationnels, un mécanisme destiné à indiquer, pour chaque occurence d'un objet, le schéma par lequel il à été introduit dans le schéma global.

Par contre, dans l'autre approche, le programme interactif posera la question au concepteur de S2 : "Avez vous été assez précis ?" et au concepteur de S1 "N'avez vous pas commis une erreur ?", de maniére à modifier un des schémas pour obtenir la même cardinalité.

6.3.2 Confidentialité

Un utilisateur du schéma S1 peut il avoir accès aux occurences d'un objet commun avec le schéma S2 mais introduit par ce dernier ?
Pour cette question, nous n'avons relevé aucune différence entre les deux approches.

I.6.4 Conclusion

Il semble raisonnable de modifier le moins possible les schémas conceptuels des utilisateurs.
Nous devons également tenir compte des performances, théoriquement il est possible d'intégrer n'importe quelle correspondance par généralisation mais dans ce cas le mapping opérationnel devra être très puissant ce qui nuirait évidement aux performances.

=> Le principal problème de l'intégration, c'est à dire la recherche des correspondances ne nous paraît pas différent quelle que soit la méthode utilisée.

La définition du mapping opérationnel utilisée jusqu'à présent peut être précisée comme suit:

Il se décompose en:

=> Le mapping opérationnel d'accès aux structures est l'ensemble des règles permettant de transformer les structures utilisées dans les requêtes destinées aux schémas conceptuels initiaux en structures utilisables dans le schéma intégré.

=> Le mapping opérationnel d'accès aux occurrences est destiné à résoudre le probléme des accès aux occurences (cfr 9.1), de plus il assure le remplissage initial de la BD intégrée avec les occurrences stockées dans les BD existantes à intégrer.
Il ne résout cependant pas le problème de confidentialité.
On peut d'ailleurs se poser la question : faut-il intégrer des bases de données pour ensuite interdire à l'utilisateur d'une de celle-ci d'avoir accès à celles de ses collègues ?

Vu les lignes précédentes, il semble possible de combiner les deux approches dans la même méthode, comme suit :

1. **recherche des correspondances**

[2. **modifications des schémas**]

3. **intégration par généralisation**

[4. **mapping opérationnel d'accès aux occurrences**]

5. **mapping opérationnel d'accès aux structures**

1+2+3(en partie)**+5** = intégration de schémas.
1+3+4+5= intégration de B.D.

L'étape deux sert notamment à résoudre les conflits de cardinalité et d'incompatibilité dans l'intégration de schémas conceptuels.

Chapitre 7: COHERENCE D'UN RESEAU DE CORRESPONDANCES

I.7.1 Définition

I.7.2 Règles de transitivité

I.7.3 Correspondances inverses et cohérentes

I.7.4 Exemple

I.7. Cohérence d'un réseau de correspondances

Avant d'intégrer des schémas ou des BD existantes, il faut d'abord avoir la certitude que le réseau de correspondances reliant les objets qu'ils contiennent est cohérent.

I.7.1 Définition

=> l'analyse de cohérence d'un réseau de correspondances se fait de la manière suivante:

Soit un circuit de N correspondances reliant N objets appartenant à N schémas différents.

On supprime C1, une des correspondances de ce circuit et on applique les règles de transitivité décrites en I.7.2 aux N-1 autres, on obtient une résultante R

Si C1 est incompatible au sens du paragraphe I.7.3 avec l'inverse de R, alors le réseau est incohérent
sinon, on recommence le processus pour toutes les correspondances du circuit et pour tous les circuits du réseau.

Cette définition s'applique bien entendu aux correspondances en général, quels que soient les objets qu'elles relient.

I.7.2 Règles de transitivité

Il s'agit de définir une règle de composition transitive pour toutes les combinaisons possibles de correspondances.
Nous avons les correspondances suivantes:
identité: id
fonction: ff
plus précis que: pp
moins précis que: mp
angle de vue différent: av
même population: sp
sous-type de: st
surtype de: ts
sur-type commun: ss

On a les règles suivantes:
y o x -> z signifie: si on a la correspondance x entre un objet a et un objet b
et la correspondance y entre b et un objet c
alors, entre a et c, on peut déduire la correspondance z.

id o id -> id
pp o id -> pp
av o id -> av
st o id -> st
ss o id -> ss

ff o id -> ff
mp o id -> mp
sp o id -> sp
ts o id -> ts

id o ff -> ff
pp o ff -> pp
av o ff -> av

ff o ff -> ff
mp o ff -> mp
sp o ff -> sp

id o pp -> pp
pp o pp -> pp
av o pp -> av
st o pp -> st
ss o pp -> ss

ff o pp -> pp
mp o pp -> av
sp o pp -> sp
ts o pp -> ts

id o mp -> mp

ff o mp -> mp

pp o mp -> av
av o mp -> av
st o mp -> st
ss o mp -> ss

mp o mp -> mp
sp o mp -> sp
ts o mp -> ts

id o av -> av
pp o av -> av
av o av -> av
st o av -> st
ss o av -> ss

ff o av -> av
mp o av -> av
sp o av -> sp
ts o av -> ts

id o sp -> sp
pp o sp -> sp
av o sp -> sp
st o sp -> st
ss o sp -> ss

mp o sp -> sp
sp o sp -> sp
ts o sp-> ts

id o st -> st
mp o st -> st
sp o st -> st
ts o st -> ss

pp o st -> st
av o st -> st
st o st -> st
ss o st -> ss

id o ts -> ts
mp o ts -> ts
sp o ts -> ts
ts o ts -> ts

pp o ts -> ts
av o ts -> ts
st o ts -> /
ss o ts -> /

id o ss -> ss
mp o ss -> ss
sp o ss -> ss
ts o ss-> /

pp o ss -> ss
av o ss -> ss
st o ss -> /
ss o ss -> ss

I.7.3 Correspondances inverses et cohérentes

On définit l'inverse d'une correspondance de la manière suivante:
A -- B signifie que la correspondance A est l'inverse de B. on a:
id -- id
ff -- ff
pp -- mp
av -- av
sp -- sp
st -- ts
ss -- ss
Bien sûr, une correspondance écrite dans un sens est équivalente à son inverse écrite dans le sens contraire.

A c= B
Signifie que la correspondance A obtenue par composition transitive d'une chaîne de correspondances est cohérente avec B, l'inverse de la correspondance supprimée dans le cycle.

id c= id, ff
ff c= id, ff
pp c= pp
mp c= mp
av c= id, ff, pp, mp, av
sp c= id, ff, pp, mp, av, sp
st c= st
ts c= ts
ss c= ss

I.7.4 Exemple

Nous donnons ici une illustration de ce problème.

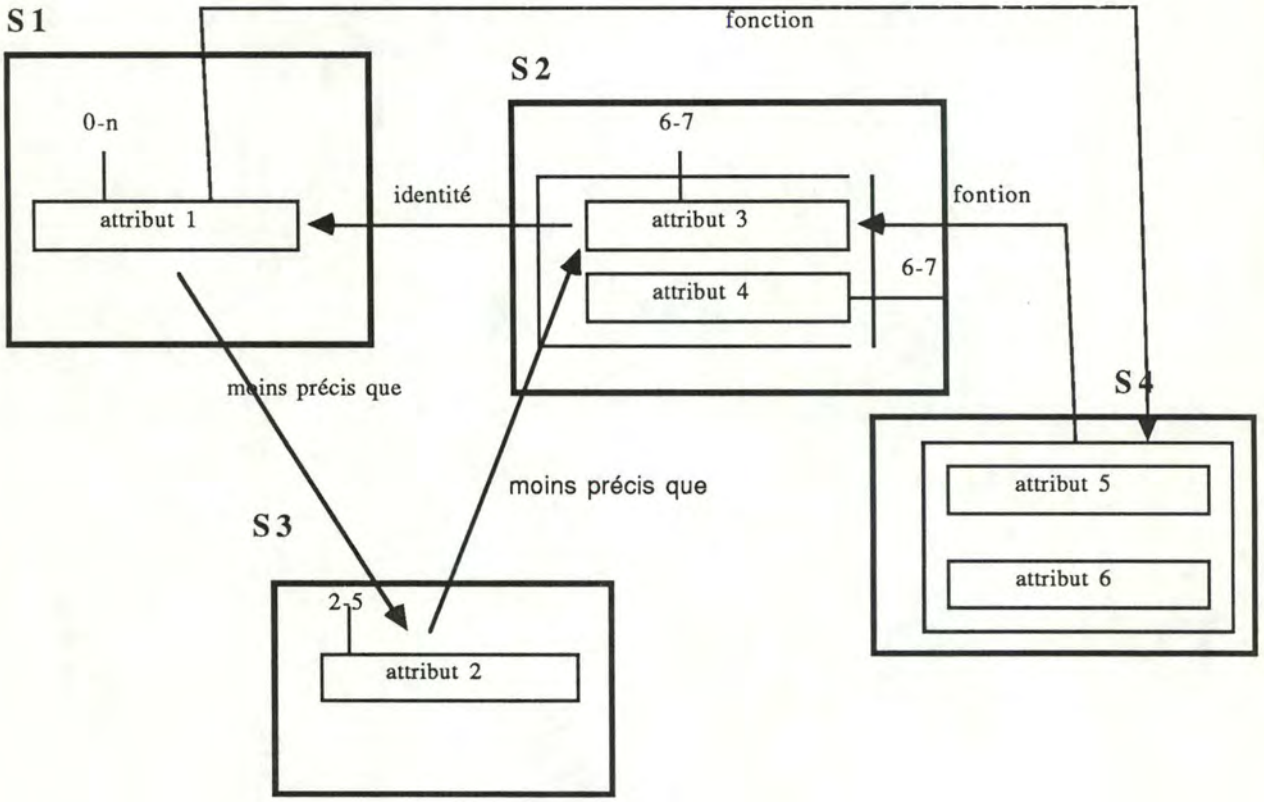


Figure 7.1: **Exemple**

Le circuit constitué des attributs 1, 2 et 3 est incohérent.
En effet, la composition id o mp -> mp dont l'inverse est pp n'est pas compatible avec mp.

Chapitre I.8: ALGORITHMES D'INTEGRATION

I.8.0. Principes de base

I.8.1. Intégration des correspondances AA

I.8.2. Intégration des correspondances EE

I.8.3. Intégration des correspondances RR

I.8.4. Intégration des correspondances EA

I.8. ALGORITHMES D' INTEGRATION

Ce chapitre présentera une série d'algorithmes destinés à automatiser au maximum l'intégration de bases de données existantes. Cependant, ces algorithmes contiennent des éléments intteractifs de manière à raffiner le principe de généralisation et donc de réduire la redondance de la base de donnée intégrée. La méthode exposée se base sur un réseau de correspondances cohérent et le plus complet possible.

I.8.1 Principes de base

Comme nous l'avons vu précédemment, dans le cas d'intégration de bases de données existantes, nous pouvons faire deux hypothèses : d'une part il est impossible de modifier les schémas, d'autre part les schémas sont supposés corrects car ils ont fait leurs preuves.
La méthode générale d'intégration sera donc la généralisation. Par généralisation, nous entendons l'utilisation de la structure ou cardinalité la plus générale. Il ne faut pas confondre avec la notion de généralisation/spécialisation utilisée pour les structures type/sous-type.
Le principal avantage de cette méthode est qu'elle peut être entièrement automatisée, tandis que son défaut est qu'elle peut introduire de la redondance dans le schèma intégré.

Distinguons d'abord les trois correspondances que nous nommerons **élémentaires**, c'est à dire Ent/Ent, relation/relation et Attr/Attr.
Lorsque nous nous trouverons devant des correspondances que nous nommerons **combinées**, par exemple Ent/Attr, le problème sera de revenir à une des correspondance élémentaire.
Nous utiliserons pour cela divers mécanismes vus ci-dessous.
Ensuite, à chaque correspondance élémentaire nous appliquerons sa règle d'intégration.
En résumé, pour chaque correspondance sus-mentionnée, nous donnerons soit une règle de transformation, soit une règle d'intégration.

Nous utilisons une méthode d'**intégration n-aire**. Néanmoins dans le but de simplifier les exemples, la plupart de ces derniers mettront en correspondance deux schémas.

Soient N schémas à intégrer.
Soient N-i objets reliés par un réseau de correspondances avec o<=i<N-1.

I.8.1. Intégration des correspondances AA

Les attributs décomposables peuvent, du point de vue de leurs composants, être considérés comme des entités. C'est pourquoi il est nécessaire de subdiviser l'étude de leur intégration.
Nous aurons d'une part les bases de toutes les règles d'intégration qui suivront, c'est à dire l'intégration des attributs non-décomposables; et d'autre part les règles d'intégrations des attributs décomposables, très semblables à celles des entités, à ceci près que les attributs dépendent d'une entité.

Dans tous les cas que nous allons détailler, il faudra en plus de l'intégration proprement dite, faire un choix sur la dénomination de l'attribut intégré.

8.1.1.Intégration d'attributs **non-décomposables** en correspondance

a] attributs identiques

C'est le cas le plus simple, il suffit de placer un des attributs dans le schéma intégré.

b] degré de précision

Explicitons d'abord le pourquoi d'une cardinalité différente alors que les attributs modélisent un même propriété d'un même objet.
Prenons la borne inférieure de la cardinalité; le même raisonnement s'appliquant à la borne supérieure. Dans l'un des schémas, soit S1 elle sera plus élevée que dans l'autre schéma à intégrer, soit S2.
Deux explications sont possibles : soit le concepteur de S1 a commis une erreur, soit celui de S2 a manqué de précision.
La possibilité d'une erreur étant exclue lors de l'intégration de BD existantes, reste le manque de précision d'un des concepteur.
L'intégration serait donc aisément réalisée par l'utilisation de la cardinalité la plus précise dans le schéma intégré. C'est-à-dire la valeur la plus élevée pour la borne inférieure et la moins élévée pour la borne supérieure.

Mais, ce raisonnement part de l'étude de la réalité. Par exemple, on peut, sans risquer une erreur, dire qu'une femme est mère de 0-20 enfants . Ce qui est évidemment plus précis que 0-n.
Raisonnons maintenant non plus sur la réalité mais sur les buts du concepteur d'un schéma. Prenons l'exemple des prénoms (cfr figure 8.1): le concepteur d'un schéma destiné à un club sportif ne **s'intéressera** qu'au premier prénom alors que le concepteur d'un schéma destiné à une école aura besoin de tous les prénoms.

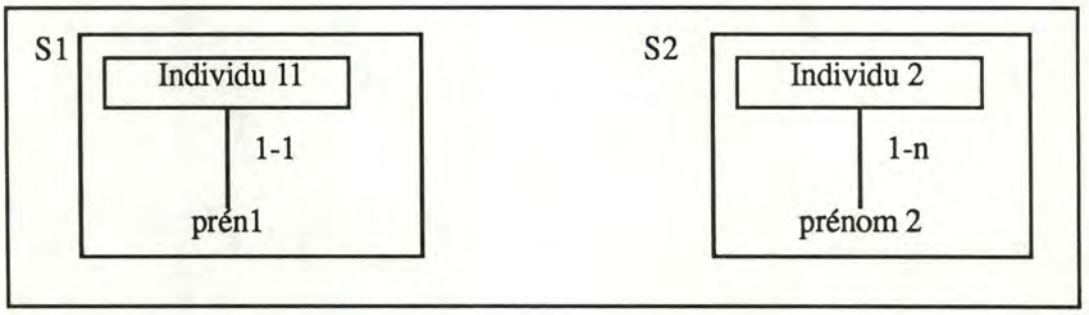


Figure 8.1: **Exemple des prénoms**

Nous ne pouvons plus dire que le premier concepteur est plus précis que le second. L'intégration par utilisation des bornes les plus contraignantes est donc impossible sous peine de perte d'information (pour le deuxième concepteur dans notre exemple).

Algorithme 8.1:

1. Nous partons de l'intervalle inclus dans tous les autres.

2. Pour tous les attributs possèdant un intervalle incluant celui-ci:
   Vos cardinalités sont elles plus larges par manque de précision ?

   Si oui, pour tous,nous prenons l'intervalle le plus petit et STOP.

   Si au moins un non , nous passons à l'intervalle englobant le précédant.

3. Retour au point 2.

De plus:

Si au moins un des attributs à intégrer est de type réel, alors nous procédons par généralisation, c'est-à-dire que dans ce cas, nous conservons le type réel dans le schéma intégré

c] attribut fonction d'un ensemble d'autres attributs

On introduit dans le schéma intégré un des (ensembles d') attributs.

Mais le problème n'est pas si simple, en effet, il faut examiner les cardinalités respectives de l'attribut fonction et de chaque attribut de l'ensemble.
Prenons le problème à sa base, nous avons comme paramètres en entrée de la fonction, un ensemble d'attributs. Si pour certains de ces attributs la cardinalité permet l'existence de plusieurs valeurs nous les prendrons toutes comme données de la fonction. Ceci évitant le problème de savoir quelle valeur prendre pour chaque attribut de l'ensemble.
Prenons l'exemple de l'adresse+n°téléphone qui est dans un des schéma un agrégat et dans l'autre un ensemble d'attributs: rue , n°, localité, téléphone; or téléphone a comme cardinalité 0-3. L'agrégat ne comprendra pas un seul mais de zéro à trois numéros de téléphone.
De plus, la fonction produit un certain nombre d'occurrences de l'attribut résultant, ce nombre étant bien entendu déterminé par sa cardinalité.
Donc, nous pouvons conserver telles quelles les cardinalités respectives de chaque élément de l'ensemble d'attributs et celle de l'attribut résultant.

Des n-i (i<n-1)attributs ou ensembles d'attributs à intégrer et reliés par une chaîne de correspondances "fonction" nous en conserverons un et un seul dans le schéma intégré.

d] angle de vue : les différences des Intervalles représentant les cardinalités sont non vides

d1 *intervalles disjoints*

Il ne peut s'agir d'un manque de précision de l'un des concepteurs, nous procédons donc par généralisation, c'est à dire en adoptant l'intervalle ayant comme borne inférieure la valeur minimale de l'ensemble des bornes inférieures des attributs à intégrer et comme borne supérieure la valeur maximale de l'ensemble des bornes supérieures des attributs à intégrer.
On peut s'interroger sur l'utilité pratique d'un tel cas. Existe-t-il une correspondance du point de vue sémantique entre deux attributs si leurs cardinalités sont disjointes ou, autrement dit: contradictoires.

d2 *chaîne d'intervalles non disjoints*

La méthode est identique à celle vue pour le "degré de précision" à ceci près que le manque de précision peut venir de plusieurs concepteurs.
Dès lors, si besoin est, nous ne prendrons plus l'intervalle le plus petit, mais bien un intervalle compris dans l'intersection et déterminé par les concepteurs.

Algorithme 8.2:

1. Ensemble de départ = ensembles des intervalles des attributs à intégrer.

2. Etude des intersections entre intervalles
   Sélection de l'intersection faisant intervenir le plus grand nombre d'intervalles.
   En cas d'égalité: choix arbitraire.

3. Pour les intervalles possédant cette intersection:
   3.1 La cardinalité représentée par l'intersection est elle satisfaisante ?
   Si oui pour tous: mémoriser cet intervalle et aller en 4
   Sinon: se mettre d'accord sur un intervalle dont l'intersection avec l'intersection étudiée est différente du vide, le mémoriser et aller en 4.

4. Enlever de l'ensemble des intervalles ceux impliqués dans l'intersection ci-dessus.
   Si l'ensemble résultant est $\neq$ du vide alors aller en 2
   Sinon , si tous les intervalles mémorisés sont disjoints aller en 5
   sinon ensemble des intervalles à étudier = ensembles des intervalles mémorisés et retour en 2.

5. Les intervalles mémorisés sont nécessairement disjoints, il faut donc leur appliquer la méthode vue ci-dessus et STOP.

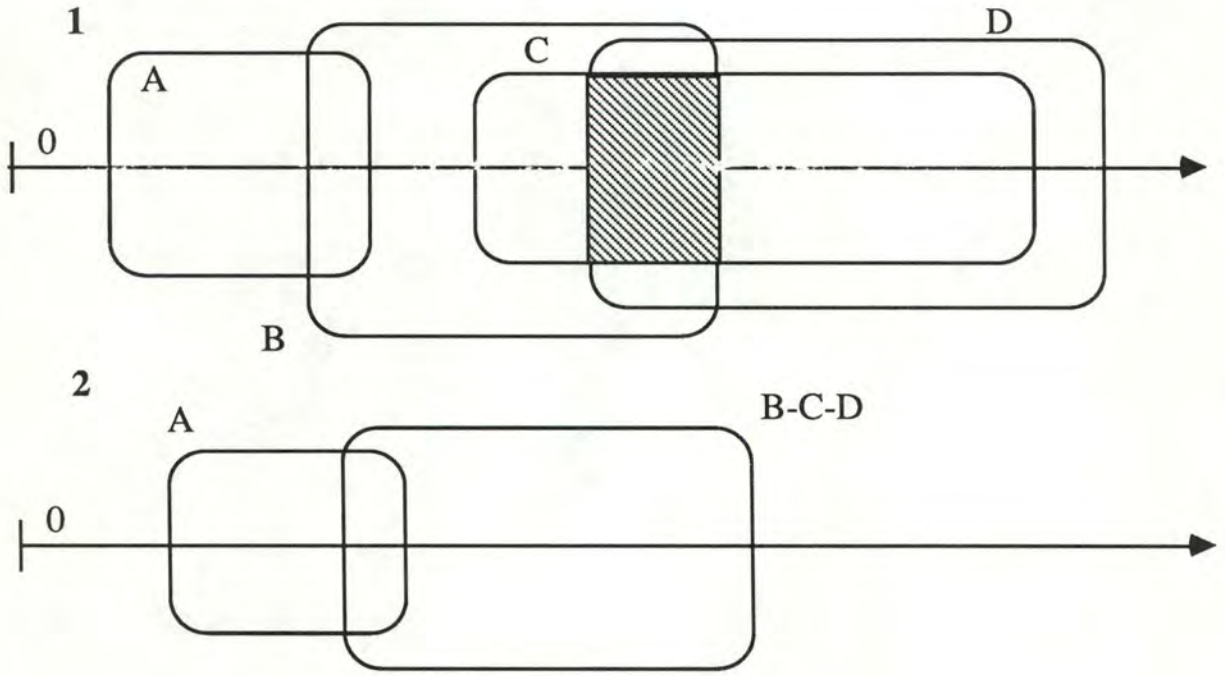


Figure 8.2: **Exemple**

L'intersection mettant en jeu le plus grand nombre d'intervalles est celle qui est hachurée.

d3 *ensembles disjoints d'intervalles non disjoints*

A l'intérieur de chaque ensemble nous intégrons par la méthode vue dans d2. Ensuite, pour chaque attribut intégré et donc possédant une cardinalité représentée par un intervalle disjoint de ceux des attributs restant, nous appliquons d1.

e] **Algorithme** général d'intégration des attributs simples **8.3**

PREMIERE ETAPE: intégration des identités

SECONDE ETAPE: intégration simultanée des correspondances "angle de vue" et "moins précis que".

Algorithme de la seconde étape:

Profondeur d'un attribut ::= longueur de la plus grande chaîne d'inclusions strictes d'intervalles pour arriver à celui de l'attribut étudié.
(cfr. exemple ci-dessous)

1. Sélectionner l'ensemble des attributs de profondeur maximum (N)

2. Pour les **x** attributs dont les cardinalités sont incluses dans un même ensemble de profondeur N-1
   Si x >= 2
   Alors les x attributs sont nécessairement reliés par une correspondance "angle de vue" que nous intégrons suivant les méthodes vuesprécédemment.
   L'attribut intégré garde évidemment la même profondeur (N).

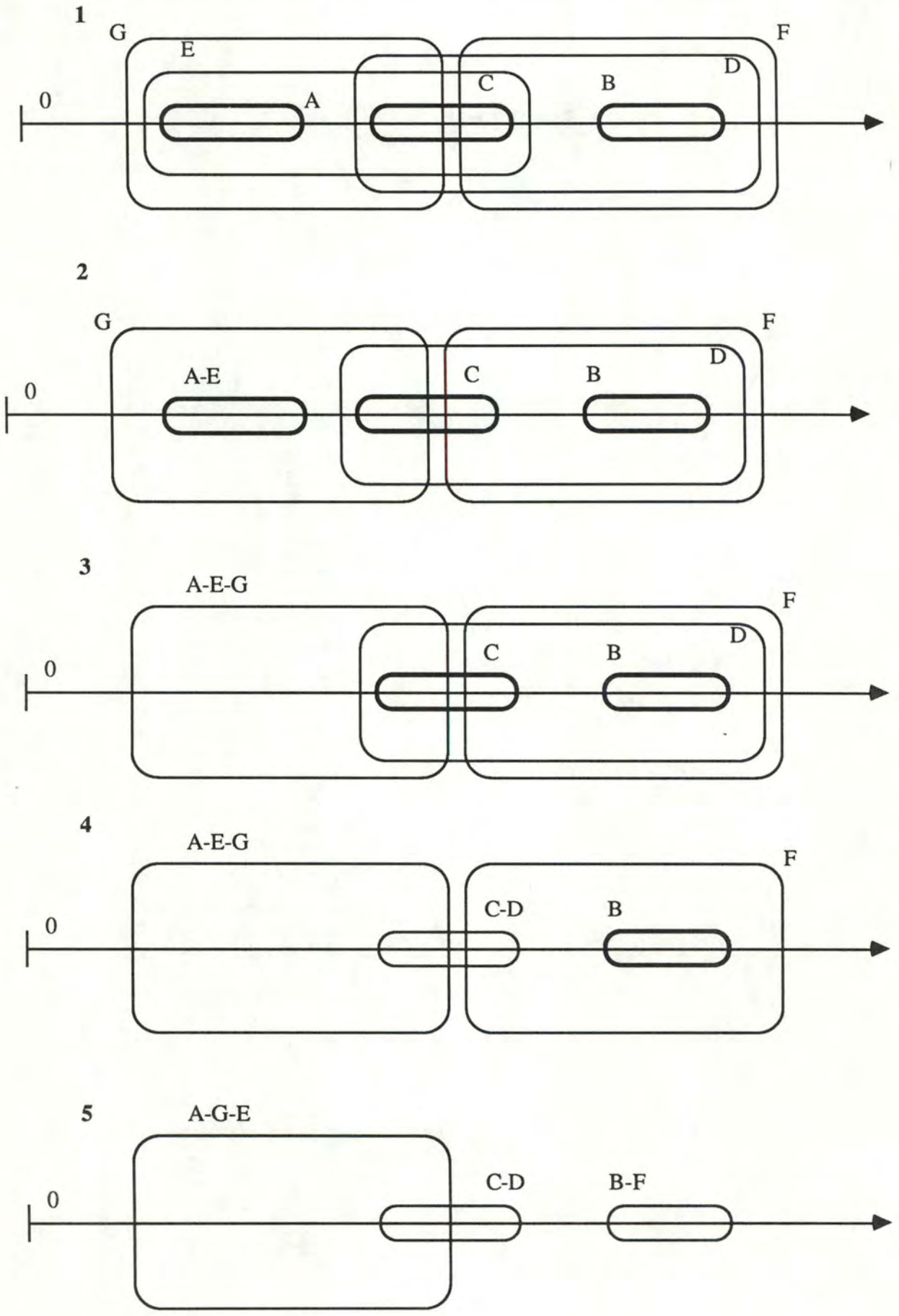
3. Les attributs de profondeur N-1 sont "moins précis que" ceux de profondeur N; nous intégrons cette correspondance par les méthodes vues précédemment.

4. Replacer l'intervalle obtenu dans l'ensemble à étudier.

5. Retour en 1 tant que l'ensemble des intervalles à étudier contient plus d'un élément.

TROISIEME ETAPE: à ce stade, il ne reste que des correspondances "fonction", que nous intégrons par les méthodes vues précédemment.

Exemple de résolution par l'algorithme de la seconde étape: figure 8.3


1
G
E
F
D
C
B
A
0
2
G
F
D
A-E
C
B
0
3
A-E-G
F
D
C
B
0
4
A-E-G
F
C-D
B
0
5
A-G-E
C-D
B-F
0

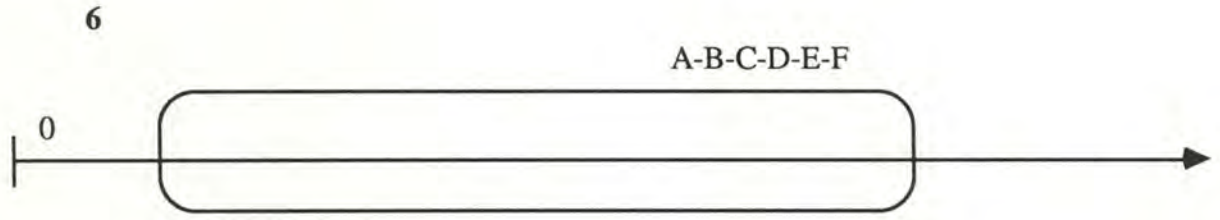


Figure 8.3: **Exemple**

Les ensembles représentent les cardinalités des attributs à intégrer.
A,B et C sont tous trois de profondeur 1. En effet, A est inclu dans E ainsi que dans G, mais E n'est pas strictement inclu dans G donc la profondeur de A n'est pas augmentée de 1.

Prenons un des ensembles de profondeur maximale, par exemple A. Par convention nous prendrons toujours l'ensemble ayant sa borne inférieure la plus proche de 0.
A est intégré avec un des ensemble de profondeur 0 (N-1) l'incluant, par exemple E.
Après mise au point entre les concepteurs concernés, il apparait que celui de E a manqué de précision et peut très bient utiliser A.
Le processus se poursuit jusqu'à obtention d'un seul intervalle.

f] critique de l'algorithme général 8.3

L'algorithme ne résout pas le problème de la cardinalité, il propose en fait une méthode de discussion entre les divers concepteurs impliqués dans un conflit. De plus il travaille dans un réseau de correspondances bien précis, mais ne s'occupe pas de savoir par quels schémas elles passent ou ne passent pas.
Par la suite, c'est à dire lors de l'intégration des entités ou relations il faudra tenir compte pour le schéma intégré d'une possible occurrence vide d'un attribut s'il n'est pas présent dans un des objets (entités ou relations) en correspondance.

8.1.2. Intégration d'attributs **décomposables** en correspondance

L'information nécessaire consiste en une donnée sémantique: les deux attributs en correspondances modélisent-ils la même propriété d'un objet? De façon différente, plus ou moins précise peu importe. L'intégration des attributs complexes se confond avec celle des entités auxquelles ils se rattachent. Nous verrons ce processus dans le paragraphe suivant.

Le seul point intéressant est l'intégration des cardinalités des attributs complexes, en effet le problème des composants se règlera de façon récursive jusqu'à l'intégration de correspondances entre attributs simples.

Les cardinalités des attributs décomposables en correspondance sont intégrées de la même manière que celles d'attributs non décomposables

I.8.2 Intégration des correspondances EE

Dans tous les cas que nous allons détailler il faudra, en plus de l'intégration proprement dite, faire un choix sur la dénomination de l'entité intégrée.

Le détail des correspondances vues dans le chapitre 5 n'est pas nécessaire lors de l'intégration. En effet, tous les critères concernant le nombre d'attributs, leur cardinalité,... peuvent être déduits des correspondances entre les attributs des entités à intégrer. Seul les notions sémantiques devront être fournies, c'est à dire: les populations des entités sont elles équivalentes, incluses ou seulement non disjointes ?

Nous allons étudier ces trois cas séparément, ensuite nous verrons un algorithme général.

I. Si les populations des entités modélisent un même ensemble d'objets du monde réel.

Données:

1. Correspondances précises entre les attributs simples des entités à intégrer.

Dans la suite nous parlerons de correspondances entre attributs complexes, or elles ne sont pas nécessaires en tant que données. En effet, elles sont toujours déductibles des correspondances entre leurs attributs composants. Pour effectuer cette déduction, il suffit d'utiliser les définitions des correspondances AA vues dans le chapitre I.5.

Nous utilisons également des correspondances entre attributs simples et décomposables. Nous n'avons pas défini ces dernières dans le chapitre I.4 car elles sont inutiles du point de vue de l'utilisateur, et leur utilisation par les algorithmes est totalement transparente. Elles sont déduites des correspondances entre attributs simples de la manière suivante:

a - On part de deux attributs simples reliés par une correspondance.
b - Si les pères respectifs des deux attributs sont des entités, STOP.
c - Pour chacun des deux attributs: on passe à son père, sauf si celui-ci est une entité.
d - On place une correspondance globale entre les deux attributs courants.

Règles:

1. Pour tout niveau, on intègre les attributs dont les pères respectifs ont été intégrés ensemble.

2. Le cas d'un attribut agrégat relié par une correspondance fonction à des composants d'un attribut complexe est particulier.
En effet, l'agrégat est une autre représentation d'un attribut complexe, dès lors, il joue un double rôle: d'une part celui de ses composants reliés par des correspondances fonction et d'autre part, celui d'attribut complexe étant en correpondance avec l'attribut complexe duquel dépendent les composants cités plus haut.

Un agrégat doit être à la fois le point de départ d'une correspondance "fonction" avec les composants de l'attribut complexe et en correspondance avec l'attribut complexe proprement dit.

3. Pour toutes correspondances partant d'un attribut complexe, on intègre la plus élevée dans l'arbre des attributs et on marque celles de niveau inférieur. L'intégration d'un attribut complexe revient en fait à intégrer sa cardinalité.

4. Si pour un attribut on trouve une correspondance marquée (règle 3), cela signifie qu'il existe dans l'autre schéma un attribut en correspondance et qu'on n'intègre pas cette correspondance car cela a déjà été effectué dans une branche supérieure de l'arbre des attributs.

5. Si un attribut n'a pas de correspondant dans au moins une des entités à intégrer, on prévoit la possibilité d'une occurrence vide **sauf** si pour la **même** raison un de ses ancètres est déjà facultatif dans le schéma intégré.

Algorithme 8.4:

1. Si les populations des entités sont identiques
   Alors
      1.0 Si l'ensemble des attributs non-visités est différent du vide, ller en 1.1
          Sinon aller en 1.6
      1.1 On se positionne sur un des attributs non visités du niveau le plus haut.
      1.2 On intègre cet attribut avec ses correspondants.
          Pour ce faire nous utilisons l' algorithme d'intégration d'attributs simples et complexes.

1.3 On marque les attributs intégrés ainsi que toutes les correspondances de niveau inférieur à celle que nous venons d'intégrer dont ils dépendent; sauf si la correspondance de niveau inférieur est une fonction.

1.4 Retour en 1.0.

1.6 Choisir une dénomination pour cet objet.

Algorithme 8.5 d'intégration des attributs simples et complexes

1. Détermination d'une possible occurrence vide dans le schéma intégré, pour ce faire on considère le réseau de correspondances tant marquées que non marquées. La méthode est classique, une occurrence vide dans le schéma intégré peut se présenter si le réseau de correspondances ne passe pas par au moins un des arbres d'attributs à intégrer, exception faite de la règle 5.

2. On élimine les correspondances marquées appartenant au réseau étudié.

3. On utilise, suivant les cas les algorithmes vus dans le paragraphe traitant de l'intégration des attributs de plus si la réponse au point 1 est positive la cardinalité intégrée est modifiée en conséquence.

II. Si les classes d'objets modélisés par les populations des entités sont incluses l'une dans l'autre.(type/sous-type)

Soient les graphes sans boucles formés par les correspondances type/sous-type entre les entités des schémas externes:
On construit une structure dans laquelle toutes les correspondances T/S-T sont représentées par des relations IS-A.
on parcourt le graphe en largeur d'abord en commançant par les feuilles.
On considère toutes les entités du niveau le plus profond non encore parcouru et on intègre les attributs par la règle suivante:

On étudie successivement tous les ancètres directs des sous-types considérés. Si un attribut de l'ancètre est en correspondance avec un attribut d'au moins un sous-type, on intègre ces attribut selon les règles vues et on place l'attribut intégré dans le type.

Si certains sous-types ne possédaient pas l'attribut, c'est par un manque d'intérêt du concepteur de leur schéma; dans ce cas, on doit prévoir la possibilité d'une valeur vide de l'attribut intégré ainsi qu'une projection , au sens de l'algèbre relationnelle, rendant à chaque sous-type ses attributs.

Algorithme 8.6:

0. Si les classes d'objets modélisés par les populations des entités sont incluses l'une dans l'autre.(type/sous-type)

Au départ, toutes les correspondances T/S-T sont non-marquées.

On enlève du graphe toutes les correspondances qui peuvent être obtenues par composition de plusieurs autres correspondances.

Niveau courant = niveau le plus profond.

1. Tant qu'il existe une entité point de départ d'une correspondance IS-A.

1.1 Choix d'un des pères d'une entité du niveau le plus profond.

1.2 Intégration des attributs du type et des sous-types.

1.3 On place dans le schéma intégré la structure obtenue.*

1.4 Marquage de toutes les correspondances IS-A aboutissant au père.

Algorithme 8.7 du point 1.2:

1. On considère l'arbre composé par les attributs du père.

2. On élimine les correspondances entre attributs des sous-types, qui ne passent pas par un attribut du type.

3. Pour chacun de ces attributs.

3.1 On l'intègre avec les attributs de ses sous-types avec lesquelsl est en correspondance.

3.2 On place l'attribut intégré dans le père du schéma intégré.

4. On n'intègre pas les correspondances entre attributs appartenant aux sous-types uniquement.

III. Si les classes d'objets modélisés par les populations des entités ne sont pas identiques et sont incluses dans une classe plus large.

Le type placé dans le schéma intégré est une entité fictive dont la population est supposée être égale à l'union des populations des sous-types.

Lors de l'intégration, nous créerons dans le schéma intégré une structure type/sous-types.(*)

L'entité représentant le type possédera comme attributs les attributs des sous-types dont le réseau des correspondances passe par **tous** les sous-types. Les autres ne seront pas intégrés.

Algorithme général d'intégration des entités **8.8:**

0. Soit un réseau de correspondances reliant N entités
1. Intégration des correspondances impliquant une même population.
2. Intégration des correspondances "sous-type de".
3. Intégration des correspondances "sous-type/sous-type".

(*) Une entité (ou association) peut avoir déja fait l'objet d'une intégration car elle faisait l'objet d'une autre correspondance que l'on a traité précédement. Cela signifie qu'il existe déja dans le schéma intégré une occurrence de cet objet.
Lors de l'intégration courante il ne faut donc pas en créer une nouvelle mais rattacher à l'ancienne toutes les associations, sous-type, type ... que l'on serait amené à créer.

Exemples de la méthode générale d'intégration des entités dans le cas de populations modélisant les mêmes ensembles d'objets du monde réels.

A

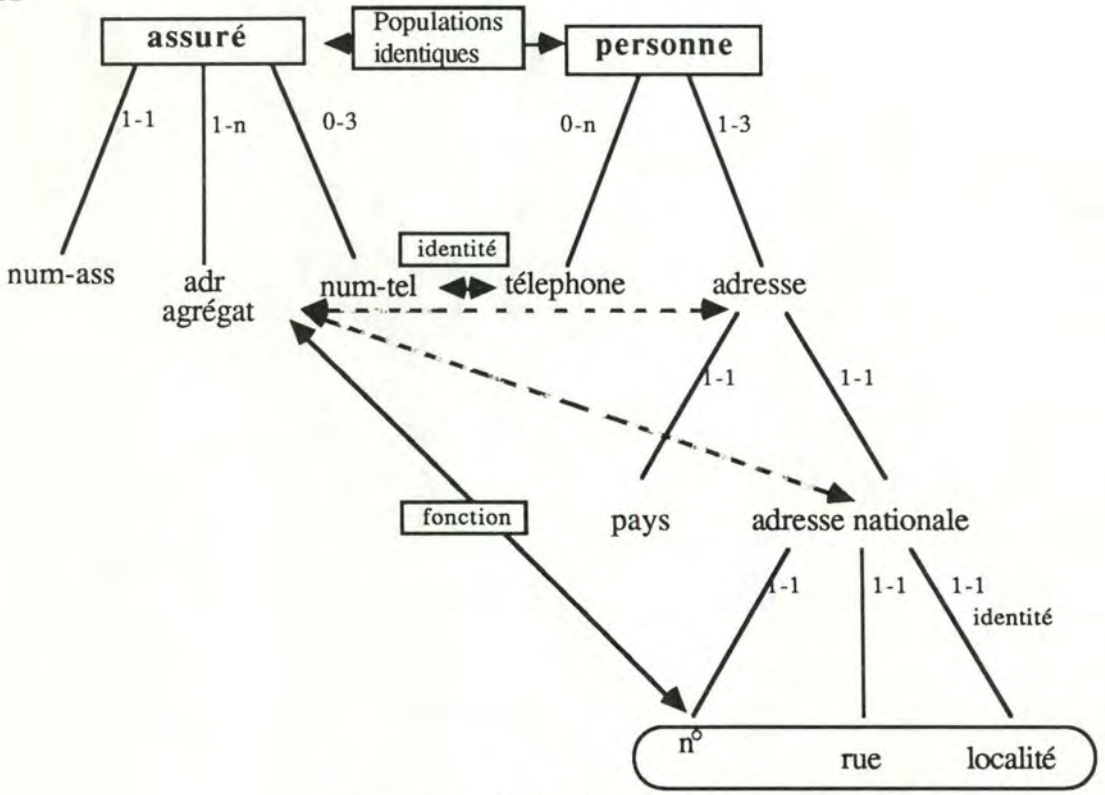


Figure 8.4: **Correspondances assuré / personne**

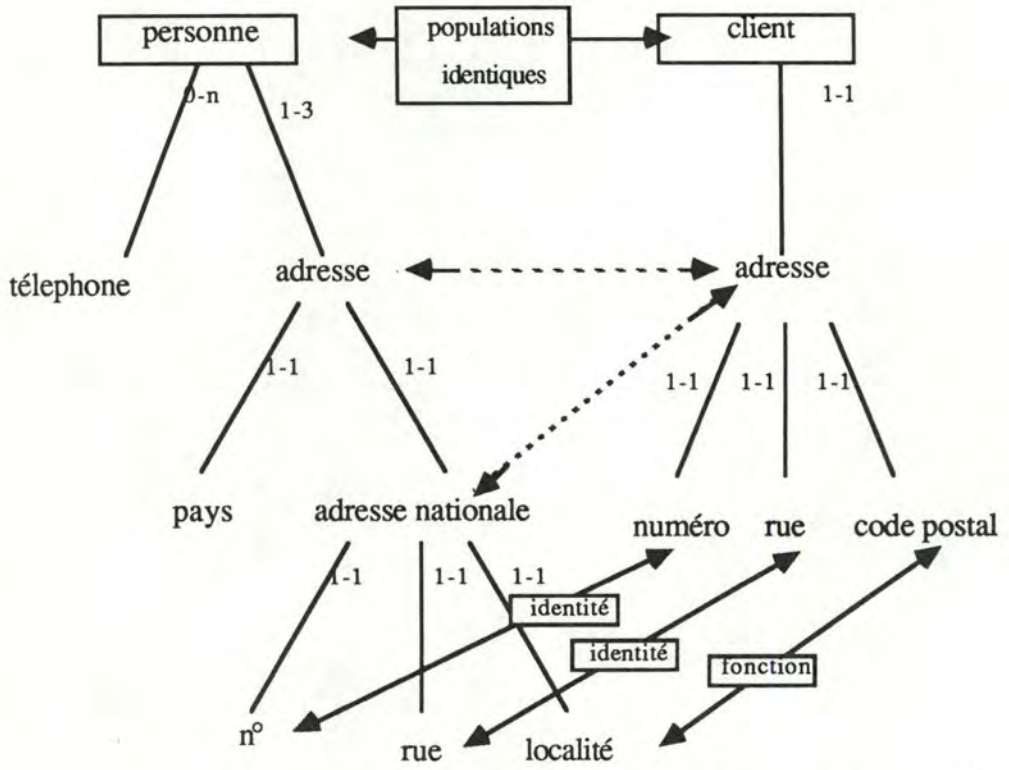


Figure 8.5: **Correspondances personne / client**

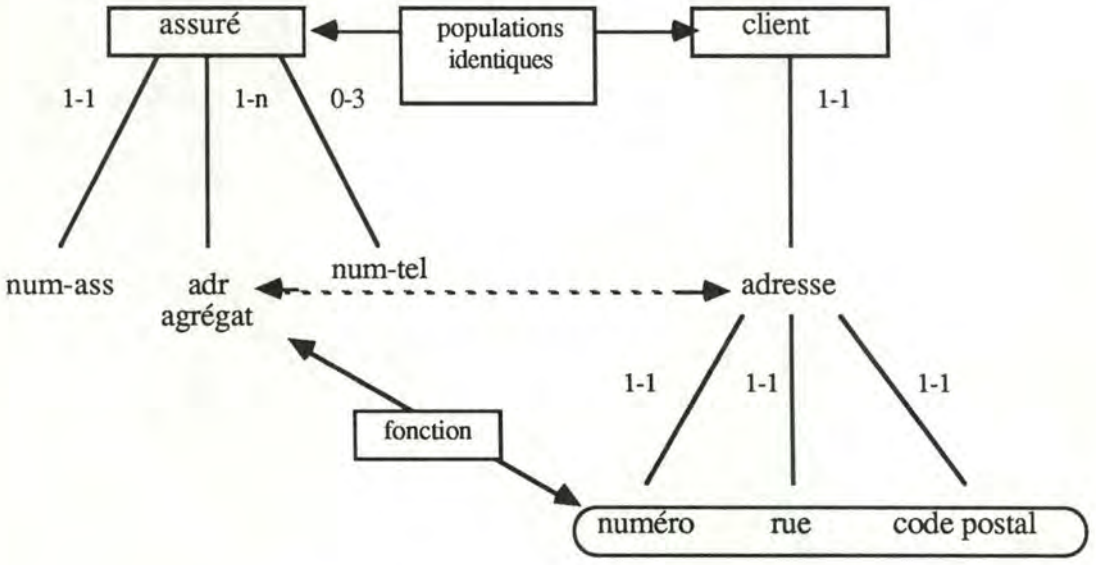


Figure 8.6: **Correspondances assuré / client**

Remarque: Les flèches en traits pointillés désignent des correspondances qu'il est possible de déduire, mais qui seront utilisées dans les algorithmes ci-dessous.
L'inscription "populations identiques" apposée sur certaines flèches est l'abréviation de "Les ensembles des objets modélisés par la population des entités sont identiques".

Les trois schémas ci-dessus expriment les correspondances nécessaires entre 3 BD à intégrer pour pouvoir utiliser nos algorithmes.
Nous intégrons les attributs les plus hauts dans l'arbre, c'est à dire num-tel/téléphone, num-ass et adresse/adr-agrégat/adresse.
Les deux premiers ne posent pas de difficultés vu qu'ils sont non-décomposables. Les correspondances passant par adresse.personne/ adr-agrégat/ adresse.client sont intégrées au niveau des cardinalités, après discussion, le concepteur de assuré admet avoir manqué de précision d'une part et d'autre part l'attribut est obligatoire dans tous les schémas donc pas besoin de prévoir l'occurrence vide.
Les correspondances du réseau ci-dessus sont éliminées et celles partant des mêmes attributs mais partant vers un niveau inférieur sont marquées (sauf les fonctions) ainsi que les attributs intégrés.
Nous passons donc au deuxième niveau.
L'attribut pays ne se trouve que dans un des schémas et son père était en correspondance avec tous les autres donc (règle 5) il faut prévoir pour pays la possibilité d'une occurrence vide dans le schéma intégré.
Adresse nationale est en correspondance avec des attributs de tous les autres schémas donc pas d'occurrence vide dans le schéma intégré.

Mais toutes ces correspondances sont marquées donc on n'intègre pas leur cardinalité avec celles des attributs correspondants.
Le reste des correspondances se trouve entre attributs simples et sont intégrées par l'algorithme 8.3.

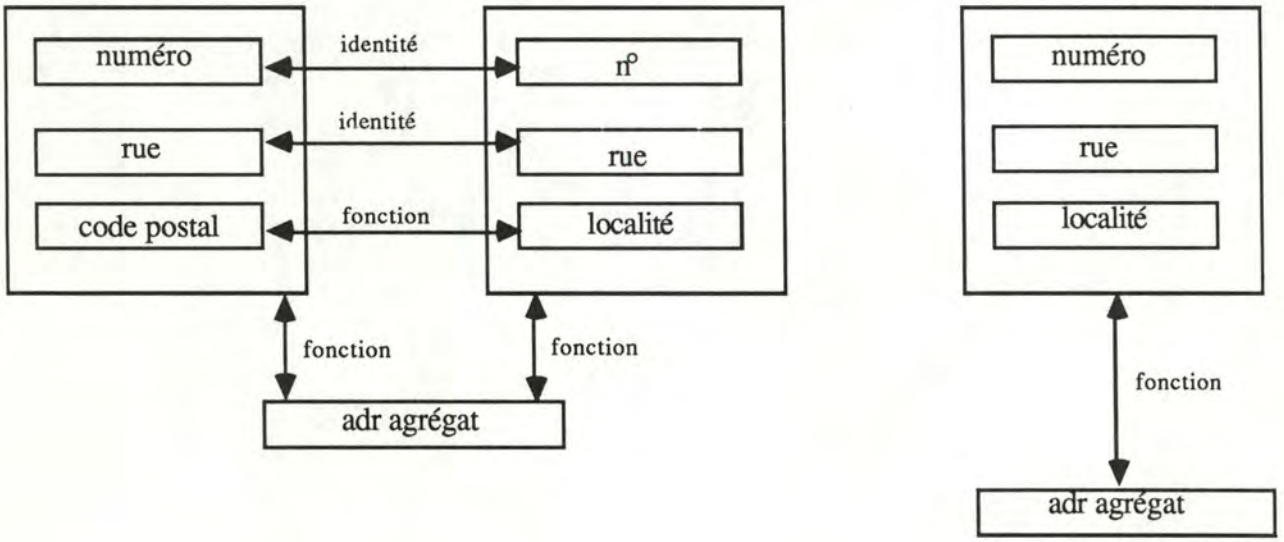


Figure 8.7 et 8.8: **Correspondances à intégrer**

Voici le schéma intégré:

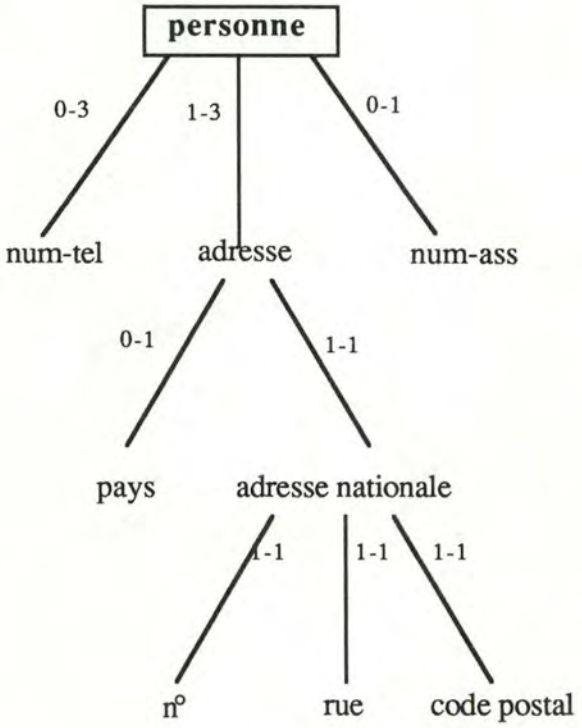


Figure 8.9: **Résultat de l'intégration**

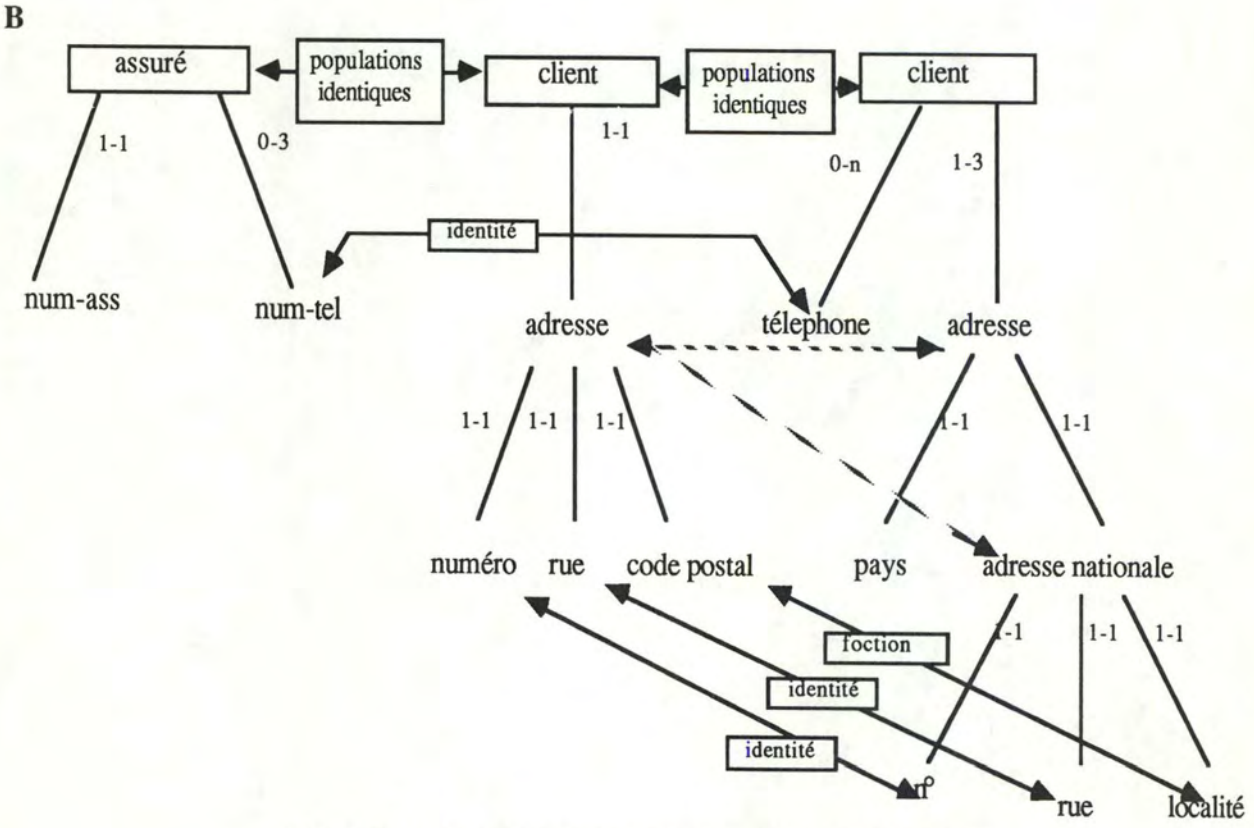


Figure 8.10: **Correspondances assuré / client / client**

Lorsque nous intégrerons les attributs adresse nous devrons tenir compte dans le schéma intégré d'une possible occurrence vide car il ne se trouve pas dans l'entité assuré. De plus la correspondance entre adresse et adresse nationale sera marquée en vertu de la règle 3.
Lors de l'intégration de l'attribut adresse nationale nous trouvons une correspondance marquée vers une des entité et aucune correspondance vers l'autre. Dés lors, par la règle 5 et le fait que son ancètre direct a déja été intégré avec possibilité d'occurrence vide pour la même raison, à savoir l'absence de correspondance vers l'entité assuré, il ne faut plus en tenir compte pour adresse nationale.

Voici le schéma intégré:

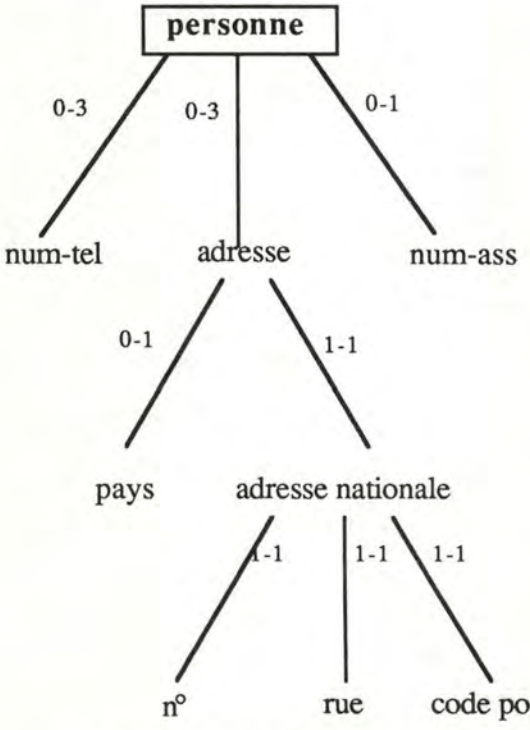


Figure 8.11: **Résultat de l'intégration**

Exemple de la méthode générale d'intégration des entités dans le cas de correspondances utilisant la notion de sous-type.

Les schémas de l'exemple A restent valables à l'exeption des correspondances entre les entités:

C

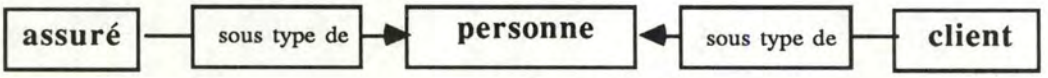


Figure 8.12

Voici le schéma intégré:

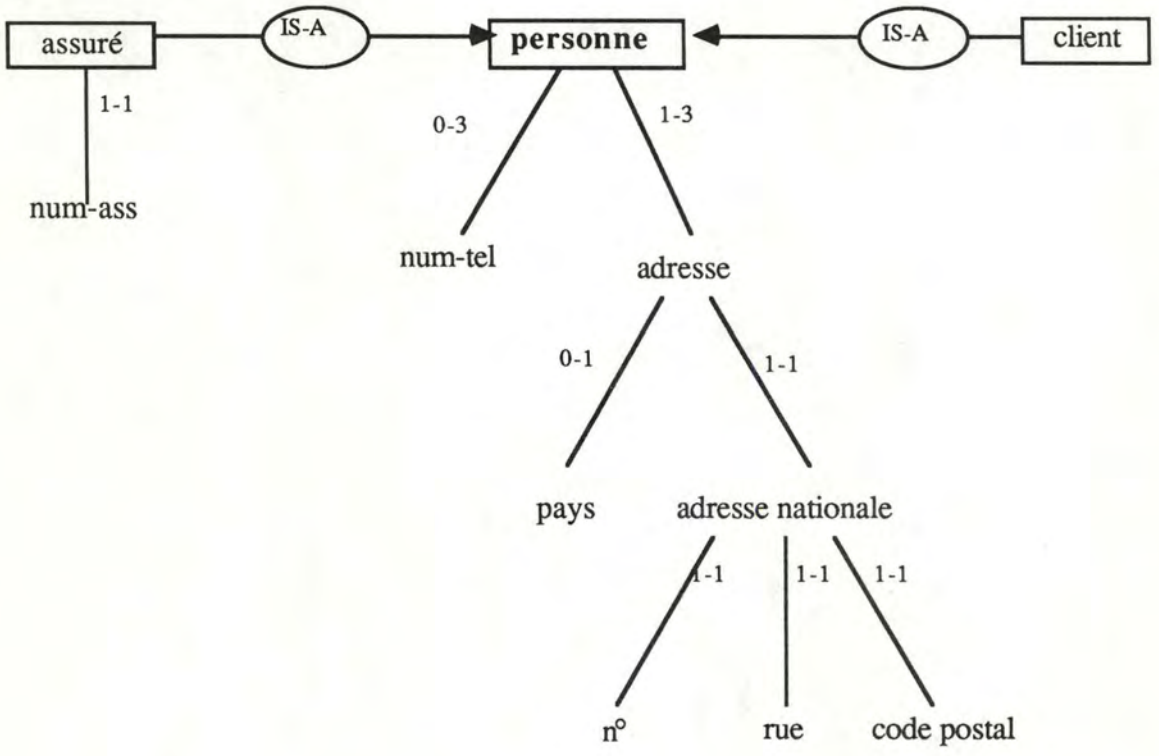


Figure 8.13: **Intégration de sous-types**

I.8.3. Intégration des correspondances RR

Données:

1. Rôles en correspondance (au moins deux)
2. Les comparaisons (inclusion, égalité, intersection vide) entre les ensembles d'interactions (sur les objets du monde réel) modélisés par les populations des relations.
3. Correspondances entre les attributs simples des relations à intégrer.

Algorithme 8.9:

1. Si la donnée 2 est l'égalité:
   Alors 1.1 Placer dans le schéma intégré une des dénominations des relations.
   1.2 Intégrer leurs attributs par la méthode vue pour les entités.
   1.3 Tous les rôles de dénomination synonyme sont intégrés en enant compte de leurs cardinaliés. Les méthodes vues pour l'intégration des cardinalités des attributs sont valables ici.
   1.4 Les rôles n'étant pas en correspondance (1.3) sont placés dans le schéma intégré en tenant compte dans la nouvelle cardinalité de la possibilité que l'entité qu'ils relient ait une occurrence vide.

2. Si la donnée 2 est l'inclusion:
   Alors 2.1 Nous transformons les associations en entités qui sont elles-mêmes reliées par des correspondances "type/sous-type".
   2.2 Nous intégrons ces identités et leurs attributs par les algorithmes adéquats.
   2.3 Nous intégrons les cardinalités de rôles ayant une dénomination synonyme.
   2.4 La cardinalité des autres est idem 1.4
   2.5 Les rôles intégrés sont transformés en associations qui elient le nouveau sur-type d'entités et les entités anciennement attachées aux relations à intégrer.
   2.6 Les cardinalités de ces nouvelles correspondances sont:
   du côté du sur-type: 1-1
   du côté de l'entité: la cardinalité du rôle intégré avant sa transformation en association.

3. Si la donnée 2 est l'inclusion dans un ensemble plus large (avec ou sans intersection vide des ensembles concernés).

Alors les relations sont transformées en entités sous-types d'un même sur-type. Pour le reste, le processus est identique à celui du point deux à l'exception du fait que le sur-type ne se trouve dans aucun schéma et est donc créé pour le schéma intégré.

4. Si la donnée 2 est l'intersection vide uniquement:

Alors les associations concernées ne sont pas intégrées car elles n'ont pas à être en correspondance.

Exemples de la méthode générale d'intégration des associations .

A

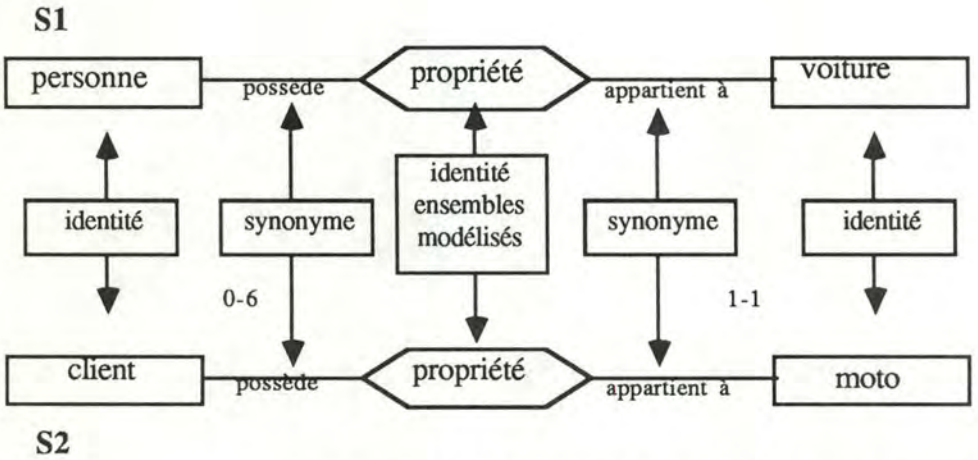
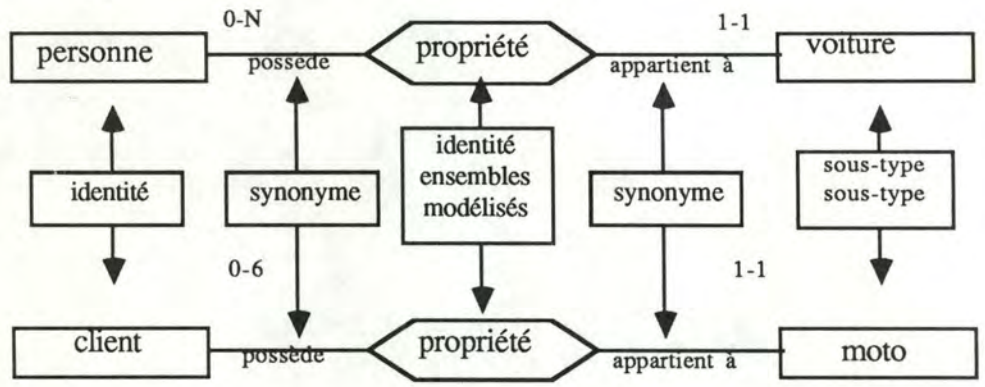


Figure 8.14: **Correspondances entre associations**

En admettant que les concepteurs se sont mis d'accord sur la cardinalité du rôle "possède", la résolution est évidente.

B

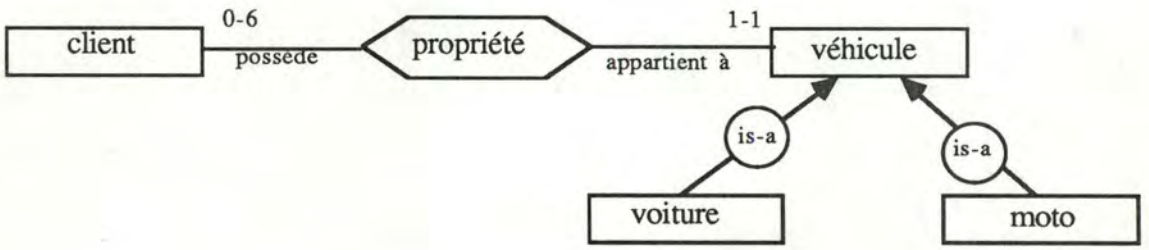
S1



S2

Figure 8.15: **Correspondances entre associations**

Notez la correspondance entre moto et voiture.
Une alternative s'offre à nous suivant les possibilités du modèle ERA utilisé. Le premier schéma est possible si la structure type/sous-type permet l'héritage des associations attachées au type.



Dans le deuxième schéma nous avons utilisé la notion de rôle multi-domaines.

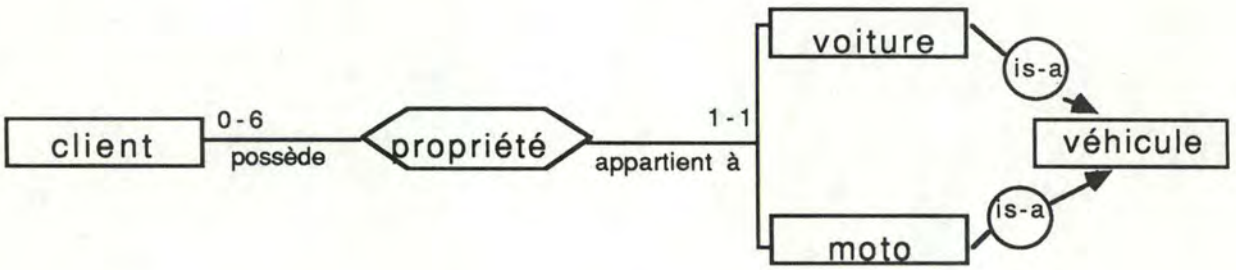


Figure 8.16: **Résolution**

8.4. Intégration des correspondances EA

Le cas se présente comme suit:

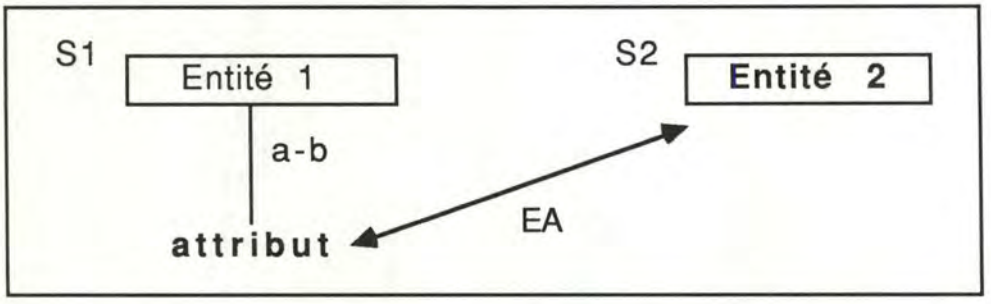


Figure 8.17: **Correspondance entité / attribut**

Le mécanisme d'intégration est de transformer l'attribut en entité et de relier cette entité par une relation à l'objet auquel elle était rattachée en tant qu'attribut.

Ce mécanisme est montré dans la figure suivante:

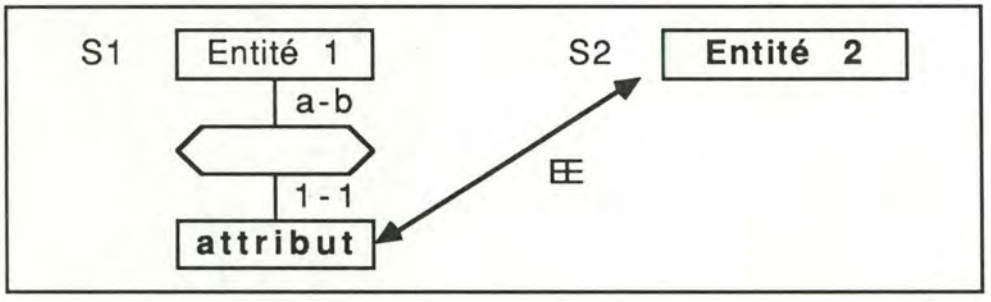


Figure 8.18: **Mecanisme d'intégration**

Une fois cette transformation effectuée nous nous retrouvons devant une correspondance EE de type "identité","angle de vue" ou "précision".

La figure suivante montre une intégration après transformation d'un cas simple:

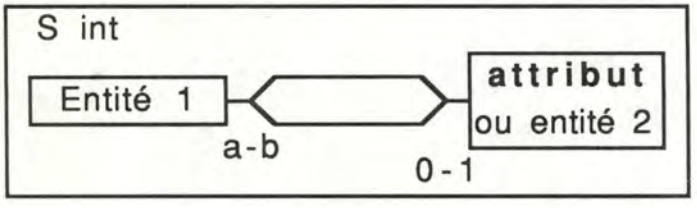


Figure 8.19

Les cardinalités de la relation doivent respecter, d'une part les cardinalités de l'attribut, ici a-b pour le premier rôle et d'autre part le fait qu'une occurrence de l'entité 2 peut exister sans être attachée par une relation à l'entité 1, c'est à dire 0-1 pour le second rôle.

En cas de relation entre un composant d'attribut complexe et une entité. Le mécanisme d'intégration est simple mais il introduit une grande complexité dans le schéma intégré.
En effet, la méthode ci dessus ne fonctionne plus du fait qu'il n'existe pas de relation entre attribut et entité. Il faudra dès lors transformer tous les attributs de niveaux supérieurs en entités. Prenons le cas d'un attribut de niveau n, il faudra donc créer n entités et n relations dans le schéma intégré.

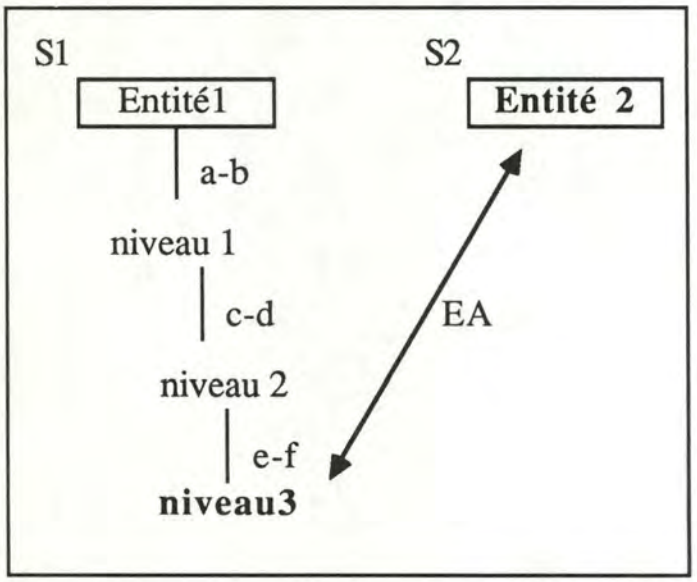


Figure 8.20

La transformation donne le schéma S1 suivant:

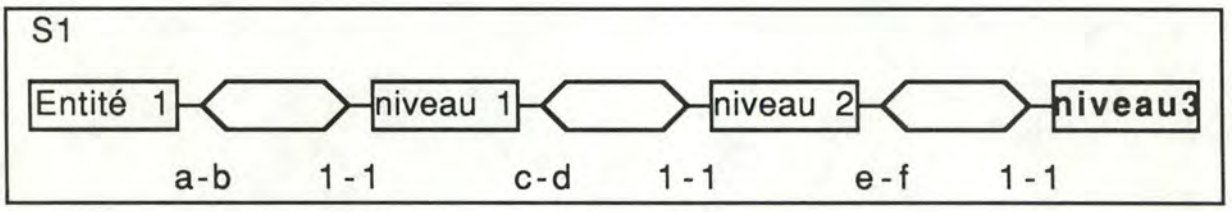


Figure 8.21

Une remarque vient à l'esprit: pourquoi avoir regroupé dans une même correspondance des cas qui s'intègrent différemment?
Car lors de l'établissement des correspondances nous avons tenu compte uniquement de la sémantique de ces dernières et non du traitement du traitement à leur appliquer. De cette manière nous nous plaçons dans l'optique de l'utilisateur pour qui le traitement des correspondances n'offre aucun intèrêt, seul leur établissement pose problème et de ce fait doit être facilité.

Chapitre I.9: MAPPING OPERATIONNEL

I.9.1. Accès aux occurrences d'un attribut

I.9.2. Conflit de cardinalité

I.9.3. Identifiant d'une entité

I.9.4. Projection

I.9.5. Confidentialité

9. MAPPING OPERATIONNEL POUR BD EXISTANTES

9.1.accès aux occurrence d'un attribut

L'attribut intégré possédant une cardinalité plus large ou au moins égale à celle de tous les attributs de départ, il faut retrouver dans l'ensemble des occurrences de l'attribut intégré celles qui ont été introduites par tel ou tel schéma.

Pour cela, nous posons deux hypothèses

1] L'ordre des valeurs des occurrences des attributs multivalués est indifférent. Si l'on accorde un ordre particulier il sagit d'une mauvaise modélisation ou d'un mauvais choix de correspondances.

2] Les ensembles de valeurs rentrés par les différents schémas sont soit égaux, soit inclus l'un dans l'autre. S'ils sont disjoints nous prenons un des ensemble au hasard. Si l'un des concepteurs n'est cependant pas d'accord avec ce procédé c'est que les attributs possédant ces ensembles disjoints n'auraient pas du être intégrés.

Le procédé consistera à ranger l'ensemble des valeurs en mettant en premier lieu les valeurs du plus petit ensemble. Puis les valeurs différentes du plus petit ensemble qui lui est supérieur et ainsi de suite. Lors de la restitution des valeurs nous prendrons simplement les x **premières** qui sont demandées.

9.2. conflit de cardinalité

Un problème se pose lorsqu'un utilisateur d'un schéma à un droit d'accès aux occurrences d'une entité qu'il n'a pas nécessairement introduite. Si cette entité possède un attribut tel que la borne inférieure de sa cardinalité est plus élevée que celle de son correspondant du schéma intégré, une violation des contraintes d'intégrité du schéma demandeur peut s'ensuivre. Il faut donc prévoir un mécanisme règlant ce type de conflit.

9.3. identifiant d'une entité

Première possibilité: l'identifiant d'une identité n'est pas le même que celui des identités en correspondance, dès lors ces attributs ne seront plus identifiants dans le schéma intégré.
Seconde possibilité: les identifiants des entités en correspondance sont tous reliés par un réseau de correspondances "identités" . L'attribut intégré peut ne pas être identifiant s'il à été intégré à partir d'indentifiants locaux à leur schéma mais qui, pris dans le contexte plus général de la base de données intégrée ont perdu cette propriété.
Exemple: plusieurs sociétés d'assurances ont décidé de fusionner leur BD. A l'intérieur de celles-ci il existe une entité client qui est identifiée par un numéro: **num-cli** de type entier supérieur à zéro.
Soit dans la BD 1 le client Dupont de num-cli = 235
Soit dans la BD 2 le client Durand de num-cli = 235
etc...
Il est évident que malgré le fait que les attributs identifiants soient en correspondance "identité" il est faux de les intégrer en considérant le résultat comme identifiant.
Si par contre l'identifiant avait été le numéro de carte d'identité celui-ci aurait été conservé dans la BD intégrée. Par ce que le numéro de carte d'identité est identifiant dans un champ englobant toutes les BD à intégrer.
En résumé, il suffit qu'une des BD à intégrer n'appartienne pas à ce champ pour enlever à l'attribut intégré sa propriété identifiante.

Résolution:
1] Identifiant commun et de portée locale pour au moins un des schémas à intégrer: on crée un sous-ensemble des entités à intégrer. Les composants de ce sous-ensemble possèdent un attribut répondant au cas 3].
A l'intérieur de chaque sous ensemble l'attribut est identifiant, il faut donc le complèter avec un minimum d'information pour le rendre identifiant dans l'union de tous les sous-ensembles. Un des moyens est de le compléter par la liste des noms des schémas pour lesquels il est identifiant.Le mapping, lors d'une demande d'accès , complètera l'identifiant par le nom du sous-ensemble comprenant le schéma demandeur. A partir de ce moment l'accès se fera de manière classique.

2] Les entités à intégrer sont identifiées chacune par un attribut différent: L'identifiant du schéma intégré est composé de tous les identifiants utilisés dahs les schémas initiaux.

3] Identifiant commun et de portée globale: il reste identifiant dans le schéma intégré.

Remarque: ce procédé fonctionne également pour les associations.

9.4. projection

En toute généralité, il peut exister dans le langage d'interrogation de la BD une commande permettant d'accèder à tous les attributs d'une entité sans les nommer.
Si tel est le cas, le problème est que le schéma demandeur risque de recevoir certains attributs dont il ignore l'existence. Il faut donc projeter l'ensemble des attributs de l'entité intégrée sur l'ensemble des attributs de l'entité du schéma demandeur.

9.5. confidentialité

Si un des concepteurs désire interdire l'accès à tous les attributs d'une entité ou d'une association, toute occurrence de cette dernière sera marquée par l'indicatif du schéma qui à introduit l'objet.
Nous pourrions éventuellement donner comme valeur à la marque un ensemble de nom de schémas autorisés.
Lorsque le programme de mapping verra cette marque apposée sur une occurrence de l'entité ou de la relation il refusera de communiquer les valeurs de ses attributs; sauf évidemment si le nom du schéma demandeur est compris dans l'ensemble représenté par la marque.

Facultés universitaires N.D. de la Paix Namur

Institut d'informatique

Intégration de schémas conceptuels ERA

M.Célis & E.Deudon

TOME 2

Promoteur: F.Bodart

Mémoire présenté en vue de l'obtention
du grade de
licencié et maître en informatique

Année académique 1987-1988

SECONDE PARTIE:

IMPLEMENTATION DE LA DETECTION DES CORRESPONDANCES

Chapitre II.1: INTRODUCTION

II.1.INTRODUCTION

Cette seconde partie concerne l'implémentation d'un sous-système de la méthode d'intégration.
Les trois possibilités qui nous étaient offertes quant à la partie à implémenter sont:

1. la détection des correspondances inter-schémas
2. l'intégration proprement dite
3. la réalisation des mappings opérationnels

Le second point présente peu de difficultés, nous avons décrit les algorithmes nécessaires à sa réalisation dans la première partie de ce travail.
Le troisième point est un domaine qui a déjà fait l'objet d'études et est relativement bien connu.
Nous avons choisi le premier point car d'une part , peu de propositions existent actuellement pour remplir cette tâche et d'autre part , elle représente pour les concepteurs la partie la plus longue et fastidieuse du cycle d'intégration de schémas.
Comme nous l'avons expliqué précédemment, la détection des correspondances fait largement appel à des notions sémantiques que seul un être humain peut analyser. Ceci nécessite un dialogue important avec l'utilisateur, notre programme sera donc essentiellement orienté interactif.

Après avoir choisi la partie à implémenter, nous avons défini ses spécifications fonctionnelles. Nous présentons également quelques outils utilisés par notre programme. Ce point sera développé dans le chapitre II.3.

Dans un deuxième temps, nous avons recherché un système capable d'implémenter nos spécifications, notre choix se porta sur MS-WINDOWS (1).

Dans un troisième temps, nous nous sommes familiarisés avec ce système. Cette étape a révélé les exigences de MS-WINDOWS, c'est- à- dire les difficultés de concevoir proprement un programme interactif.
Dans ce cadre, nous verrons qu'il est important de tenir compte de l'aspect dynamique de l'application. Ces difficultés seront développées dans le chapitre II.2.

---

(1) Windows est un environnement de travail qui fonctionne sur MS-DOS et fournit une interface utilisateur orientée graphique, des possibilités de multitasking et une indépendance hardware vis à vis du PC. [Durant, Carlson,Yao].

Nous avons essayé de mettre en oeuvre les méthodes qui nous ont été enseignées tout en tenant compte de cette nouvelle exigence.
Pour cela, nous utiliserons les concepts de schéma de la dynamique, de spécification du dialogue, et de spécification d'objets interactifs car nous pensons qu'ils peuvent apporter une aide précieuse au concepteur. Ils seront détaillés respectivement dans les chapitres II.4, II.5, et II.6.

Nous tenons à préciser que ce problème s'étendant sur toutes les étapes du cycle de conception d'un programme, il nous a été impossible d'en approfondir tous les aspects. Nous nous bornerons à quelques réflexions sur des éléments de méthode qui -nous le pensons- devraient faciliter la conception. Nous ne prétendons pas proposer une méthodologie complète, ce qui par ailleurs, nous entraînerait trop loin du sujet de ce travail.

Chapitre II.2. PROBLEMES DE CONCEPTION

II.2.1. Méthode de conception

II.2.Problèmes de conception.

Ce chapitre va d'abord se situer dans le cadre d'une méthode de conception fréquemment utilisée, ensuite il abordera les problèmes rencontrés lors de la conception de l'architecture d'un programme interactif.

II.2.1 Méthode de conception

=> Nous utiliserons dans la suite les termes de méthodes de conception traditionnelles ou classiques. Ce qui signifie: toutes les méthodes basées sur une découpe fonctionnelle par raffinement succesifs des traitements qui ont été initialement développées pour la conception de programmes non-interactifs.
Le lecteur pourra trouver des définitions précises ainsi que des exemples dans [Bergland 87].
Nous rappelons ici les principes de la variante la plus utilisée.
La décomposition est la technique du "diviser pour règner " appliquée à la programmation. C'est une approche Top-Down de résolution de problèmes. Elle procède pas à pas, par une décomposition du problème et des raffinements succesifs du programme.
Le processus de conception peut être résumé comme suit:
1) Spécifier clairement la fonction souhaitée
2) Diviser la fonction en sous-fonctions, chacune résolvant une partie du problème.
3) Connecter les sous-fonctions de manière à réexprimer la fonction sous une forme équivalente.
4) Diviser et connecter chaque sous-fonction et ce jusqu'à une profondeur qui semble raisonnable.

Problèmes

=> Généralement, on considère le dialogue comme l'ensemble des informations échangées entre l'utilisateur et la machine, il représente non seulement la structure de ces informations, mais aussi la dynamique des échanges.

Actuellement, les problèmes de dialogue avec l'utilisateur prennent de plus en plus d'importance, et donc, les opérations d'interface sont plus nombreuses et plus compliquées. L'expérience montre que les concepteurs tombent facilement dans le piège de diluer ces opérations dans une architecture classique; ce qui donne des modules fortement couplés, c'est-à-dire un flux d'informations intermodules trop important . A ce sujet, on peut consulter [Coutaz 87].

Il est donc indispensable, lors du processus de modularisation, de séparer les fonctionnalités de l'application de celles du dialogue.

En outre, les applications interactives introduisent une nouvelle dimension dans la conception: le comportement de l'utilisateur. En effet, c'est ce dernier qui décide avec un certain degré de liberté de l'enchaînement des différentes fonctionnalités de son application. Il s'agit donc là d'un aspect **dynamique** qu'il est déconseillé de représenter dans une architecture classique. Cependant, cet aspect est extrêmement important, d'où la nécessité de le prendre en compte dès les premières étapes de la conception.

Les programmes non-interactifs se satisfont généralement d'une découpe par type de traitement, le tout chapeauté par un coordinateur. Les routines de gestion d'écran se trouvant généralement dans les niveaux inférieurs.
Cela vient du fait que dans ces programmes, on accorde la priorité aux traitements. On considère que ce sont eux qui représentent la partie compliquée de l'implémentation. L'enchaînement de ceux-ci est strictement défini; nous le qualifierons de '**statique**'. Or, les méthodes traditionnelles ont été initialement développées pour construire ce genre d'applications.

Nous ne prétendons pas que ces méthodes sont incapables d'assurer une bonne conception de programmes interactifs; il est fort probable qu'une telle architecture, soigneusement développée, y réussira très bien. D'ailleurs, bien que des méthodes originales aient été développées récemment, la majorité des concepteurs continuent à utiliser les méthodes classiques. Néanmoins, nous allons proposer dans les chapitres suivants quelques idées permettant de mieux intégrer dans la phase de conception l'aspect dynamique inhérent aux applications interactives.

Nous tenterons de spécifier de manière précise un dialogue, et de réconcilier les aspects dynamique et statique de la conception.

Chapitre II.3: SPECIFICATIONS

II.3.1. Spécifications sommaires

II.3.2. Scénario

II.3.3. Description sommaire des outils

II.3.Spécifications

Ce chapitre comprend l'ensemble des éléments qui devraient constituer la base des discussions entre concepteurs et utilisateurs. Ils traduisent les désirs des utilisateurs avec un minimum de termes informatiques. Ce dossier contient des spécifications fonctionnelles, une définition du scénario qui spécifie la dynamique de l'application et une description des outils utilisés par le programme.

II.3.1.Spécifications sommaires:

Ce programme va fournir une aide à la détection des correspondances entre schémas. Il proposera d'une part une marche à suivre, et fournira d'autre part un ensemble d'outils qui aideront l'utilisateur dans cette tâche .

Entrées:
Les textes DSL de définition des schémas. Par convention, nous supposerons qu'il n'y a dans un fichier qu'un seul texte DSL. Un fichier contenant les synonymes couramment utilisés dans le domaine des schémas à intégrer.

Sorties:
La liste des correspondances détectées, exprimées selon la syntaxe SDC [cfr chapitre I.5.].
Nous nous limiterons dans le cadre de cette implémentation à une application pour la détection des correspondances précises entre entités et entre leurs attributs.
Le cheminement de la détection des correspondances est décrit dans le scénario ci-après.

II.3.2 Scénario

=> La notion de scénario que nous utiliserons ici étend celle de dialogue en y introduisant les traitements qui sont déclenchés par les informations échangées ou qui agissent sur celles-ci.

Outre le scénario que nous allons détailler, l'utilisateur doit avoir accès à tout moment à un help, à la description d'un objet sélectionné, et à l'arbre des attributs des entités sélectionnées.

0 Identifier n schémas à intégrer, entre lesquels nous allons essayer de détecter les correspondances. Nous précisons qu'un et un seul schéma DSL se trouve dans un fichier.

1.1 Le programme propose deux schémas parmi les n de manière à favoriser les détections par transitivité;

1.2 l'utilisateur choisit lui-même une paire de schémas (cfr écran n°4), en suivant ou non les propositions du programme.
Lors d'un changement de schémas, exécution du point 4.
Au moyen des correspondances détectées entre deux schémas précédemment sélectionnés, le programme essaye d'appliquer des règles de transitivité pour trouver de nouvelles propositions de correspondances.

1.3 Construction de la liste des entités des schémas sélectionnés.

2.1 Le programme construit une liste de propositions de correspondances entre entités des schémas sélectionnés. (cfr écran n°4).
Les propositions sont faites par utilisations d'outils de détection des dénominations identiques, quasi-identiques et synonymes qui sont décrits dans la suite. De plus, certaines propositions ont été effectuées par déduction transitive.

2.2 L'utilisateur a le choix entre les possibilités suivantes:
- Retourner au point 1.2.
- Sélectionner une paire d'entités appartenant respectivement aux schémas courants, en se conformant ou non aux propositions faites par le programme.
Lors de la sélection de nouvelles entités, lancement de l'algorithme de déduction des correspondances précises entre entités (qui n'a d'effet que si l'utilisateur a entré précédemment des correspondances précises entre attributs)
Construction de la liste des attributs des entités sélectionnées.
Construction d'une liste de propositions de correspondances entre attributs, et passage en 2.3.

2.3 L'utilisateur a le choix entre les possibilités suivantes:
- Retourner au point 2.2
- Retourner au point 1.2
- Introduire une correspondance simple entre les entités sélectionnées et passer en 3.1.

3.1. Au choix:
- Retour en 2.3
- Retour 2.2
- Retour en 1.2
- Sélectionner deux attributs des entités sélectionnées en se conformant ou non aux propositions du programme, passer en 3.2.

3.2 Au choix:
- Retour au point 1.1 ou suivants
- Introduire une correspondance précise entre attributs sélectionnés, passer en 3.4.

3.4 Au choix:
- Retour à un des points entre 1.1 et 3.2.

4 Le programme effectue un contrôle de cohérence du réseau de correspondances entre les n schémas pour vérifier si les nouvelles correspondances validées par les utilisateurs ne sont pas en conflit avec celles trouvées précedemment.

4.1 S'il y a lieu, on présente à l'utilisateur le réseau de correspondances incohérent (cfr écran 3).

4.2 Dans ce réseau, il en choisit une à éliminer.

4.3 Le programme teste la cohérence du nouveau réseau.

4.4 S'il reste incohérent: retour en 4.1
Sinon on continue: retour en 1.3.
Il faut noter que si une correspondance est éliminée, toute celles déduites par transitivité le seront également.

Remarquons également que l'utilisateur peut quitter le programme à tout moment où il a un choix à faire.

II.3.4 Description sommaire des outils utilisés:

La liste de ces outils n'est pas exhaustive, nous avons sélectionné ceux qui nous semblaient les plus utiles et réalisables. Nous rappelons que l'intégration peut concerner des schémas modélisant différentes bases de données. Il est donc évident que ceux-ci ne respectent pas entre eux la contrainte imposée par DSL d'unicité du libellé.

1] Recherche des dénominations identiques et quasi-identiques:

Un des indices dont on dispose est la dénomination des objets constituant la BD.
Les objets ayant une dénomination identique ne posent aucun problème car il suffit de parcourir le texte DSL pour les détecter.

Nous pourrions également utiliser les dénominations quasi-identiques, comme par exemple: CLI et CLIENT. Pour qu'une dénomination soit quasi-identique à une autre , il faut qu'elles aient un noyau commun d'une taille suffisante.

Une méthode pourrait être de rechercher les abréviations comme suit:
- écrire les deux listes sous une forme syntaxique standard, par exemple en supprimant tous les séparateurs et en utilisant uniquement des majuscules.
- classer les deux listes d'objets par ordre alphabétique.
- prendre un des objets de la première liste et visionner tous les objets de la seconde liste commençant par la même lettre.
C'est-à-dire: voir si toutes les lettres du plus petit des noms se retrouvent dans le second nom d'objet, et ce dans le même ordre.
Exemple: "CMD" est une abréviation de "Commande".

2] Recherche des synonymes:

La détection des synonymes est plus complexe. Notre idée est de constituer à partir des dénominations les plus utilisées dans le domaine de l'application des listes de synonymes. Partant de deux listes de noms, le programme consultera son fichier des synonymes de manière à détecter les éventuelles paires de synonymes, qui sont les indices de possibles correspondances.

Ce fichier de synonymes pourrait être complété lors de la détection d'une nouvelle correspondance. De cette manière, il paraît raisonnable d'affirmer qu'après quelques intégrations, le fichier des synonymes sera suffisamment complet pour rendre cet outil efficace.

3] Déduction de correspondances par transitivité:

A partir d'un noyau de correspondances déja découvert, il est envisageable de déduire un ensemble de nouvelles correspondances.
Une des méthodes de déduction pourrait être basée sur la transitivité entre correspondances.
Prenons un exemple: soient trois entités A,B,C appartenant à trois schémas différents S1,S2 et S3.
Nous avons déja détecté: A identique à B et B plus précis que C lors de l'analyse de S1/S2 et S2/S3.
La transitivité permet d'affirmer que A est plus précis que C, ce qui constitue une proposition de correspondance qui sera utilisée lors de l'analyse de S1/S3.

Naturellement, quelle que soit la méthode de détection utilisée, il faut se rappeler qu'il ne s'agit que d'une aide. Toute détection devra donc être impérativement ratifiée par l'utilisateur. Nous disposons pour cela de la description informelle de la sémantique des structures définies.

4] Contrôle de la cohérence de l'ensemble des correspondances:

A chaque ajout d'un ensemble de nouvelles correspondances entre deux schémas; il est nécessaire de contrôler la cohérence du réseau de correspondances déduit.
Illustrons ce cas par un exemple:
Soient A,B,C des entités; A identique à B, B identique à C et C moins précis que A. La dernière correspondance induit une incohérence dans le réseau. Ce qui n'implique pas nécessairement que ce soit elle la fautive.
Cela montre également qu'il pourrait être utile de disposer des correspondances détaillées [ch 5 de la 1re partie].

Le contrôle de cohérence s'applique également aux correspondances entre attributs.

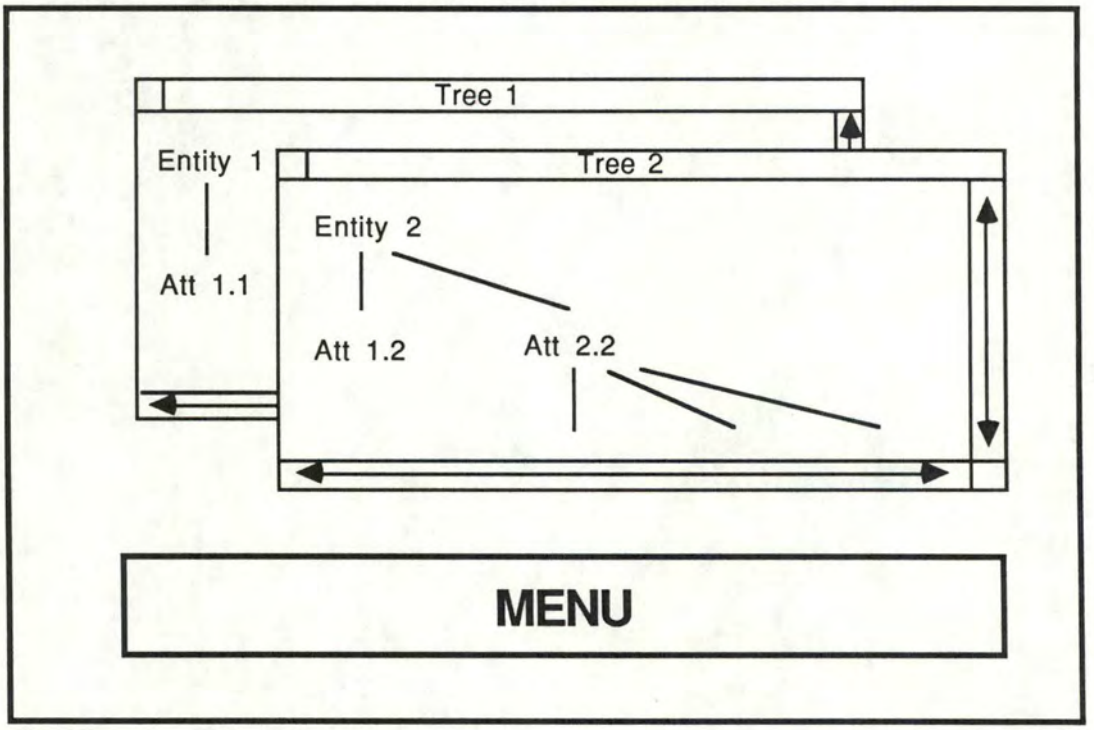
5] Déduction des correspondances entre attributs complexes:

Comme nous l'avions mentionné dans l'algorithme, il est possible de les déduire des correspondances entre attributs non décomposables. Celles-ci permettent également la déduction du type précis de la correspondance entre les entités auxquelles les attributs appartiennent.

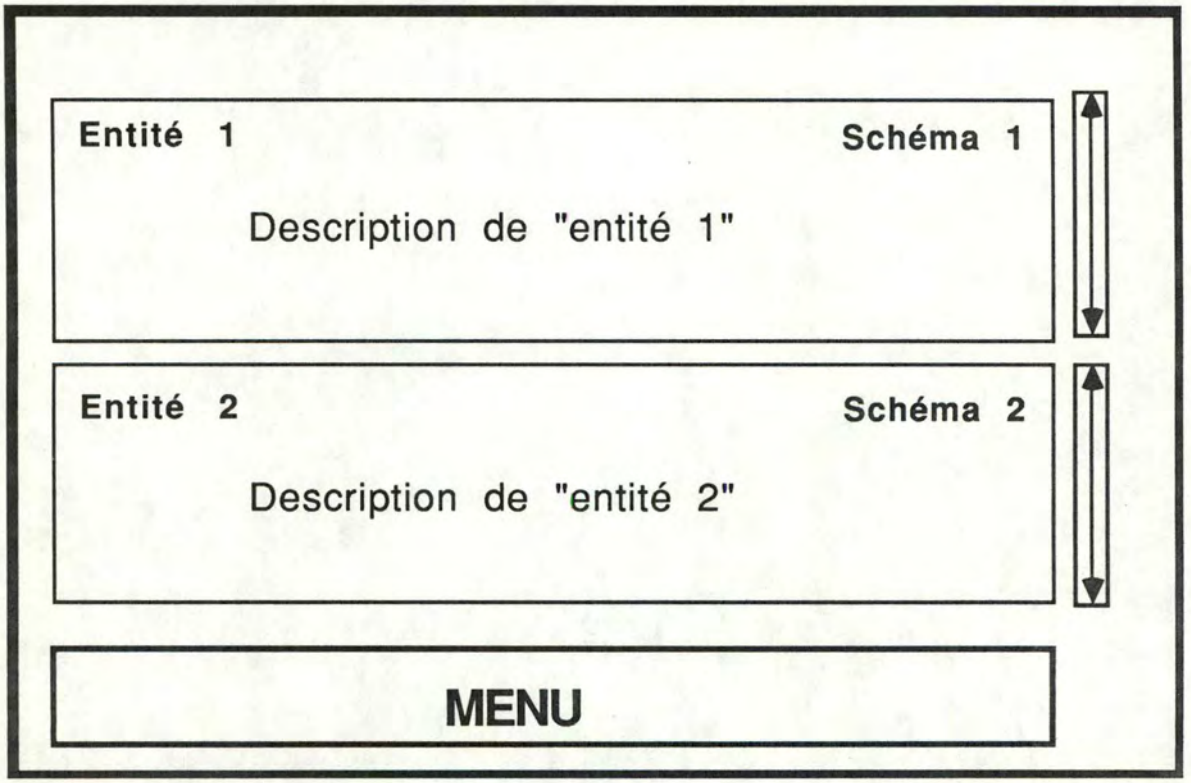
6] Visualisation du texte DSL.

Ecran 1: visualisation des attributs d'une entité ou d'une relation.
Ecran 2: visualisation de la description d'un objet.
Ecran 3: visualisation des correspondances détectées entre certains objets.
Ecran 4 : permet à l'utilisateur de choisir les deux objets sur lesquels il désire travailler.
Cet écran est utilisé en premier lieu pour savoir sur quelle paire de schémas l'utilisateur désire travailler.

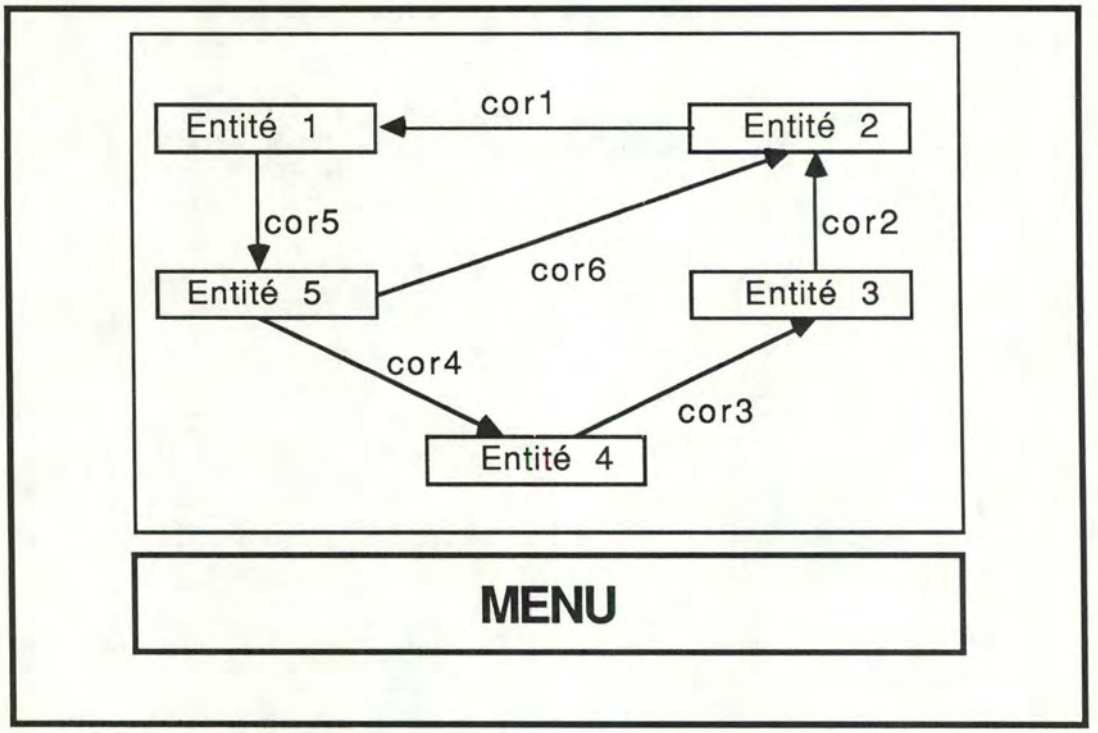
Ecran 1



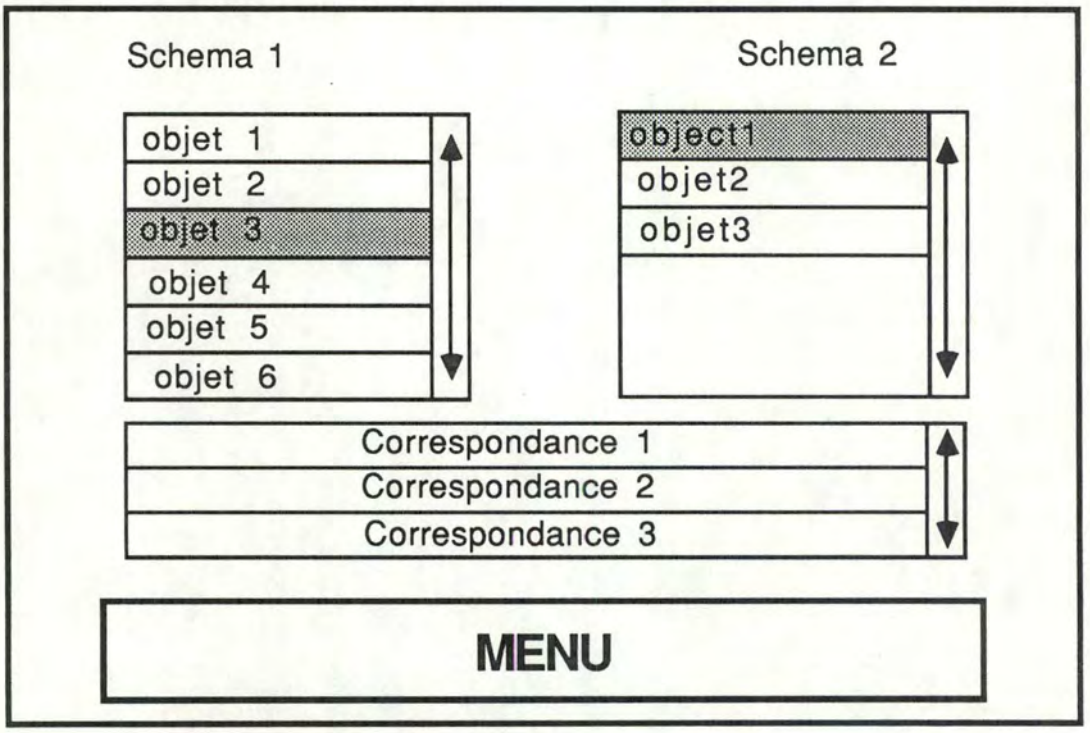
Ecran 2



Ecran 3



Ecran 4



Capitre II.4: SCHEMA DE LA DYNAMIQUE

II.4.1. Niveaux de décomposition

II.4.2. Décomposition de l'application

II.4.3. Schéma de la dynamique

II.4.4. Analyse du schéma de la dynamique

II. 4. Schéma de la dynamique

Avant de définir le dialogue, c'est-à-dire, les interactions entre l'utilisateur et la couche 'présentation' du programme, il faut insister sur le fait que si la séparation application / présentation est utile, on ne peut concevoir une interface en faisant abstraction des fonctionnalités sous-jascentes sous peine de compliquer la connexion entre l'interface et l'application. En d'autres termes, il arrive un moment où ces deux aspects se rejoignent et où on se pose la question: quels modules de traitement regrouper autour de quel module d'interface? Le diagramme de la dynamique devrait aider à ce regroupement.
Nous allons extraire des spécifications du programme une décomposition en niveaux répondants aux définitions ci-dessous. Celles-ci ont été extraites de [Bodart,Pigneur 88].
Nous présenterons ensuite, un schéma de la dynamique entre les objets résultants de cette décomposition. Ce schéma est décrit également dans [Bodart,Pigneur 88].
Nous terminerons par une analyse du diagramme en vue du regroupement des modules.

II.4.1 Niveaux de décompositions:

=> Projet: c'est la partie d'un sytème d'information qui fait l'objet d'une analyse

=> Application: c'est un traitement quasi-autonome par rapport aux autres applications d'un projet

=> Phase: c'est un traitement (manuel ou automatisable) possédant une unité spatio-temporelle d'exécution

=> Fonction: c'est le niveau élémentaire de la nomenclature des traitements. Elle est associée à un objectif et un comportement élémentaires.

Notre projet est l'intégration de schémas conceptuels; de la détection des correspondances jusqu'au mapping opérationnel.
Notre sous-système-utile représente une seule cession logique de travail, c'est à dire qu'il satisfait le critère d'unité temporelle propre à une phase.
Reste l'identification des fonctions.
Nous passerons par une étape intermédiaire durant laquelle nous détectons les fonctionnalités au sens large du terme. Ces fonctionnalités ne sont pas des actions élémentaires, nous effectuerons donc une découpe supplémentaire en fonctions. Nous donnons ci-dessous les critères utilisés dans la dernière étape.

Critère de détection des fonctions:

- agrégabilité: l'ensemble des fonctions doit remplir les objectifs associés à une phase. Une fonction correspond donc, à un sous-objectif de la phase.
- egonomie: chaque fonction d'une phase interactive doit être vue par l'utilisateur comme un service qu'il peut utiliser pour effectuer un traitement partiel et obtenir des informations, à condition de lui fournir les informations voulues, dans un état adéquat qui respecte la spécification du message associé. Les informations fournies peuvent provenir de l'utilisateur ou d'autres fonctions. L'échange d'information est indécomposable, c'est à dire que l'exécution de la fonction se déroule sans interactions avec l'utilisateur, une fois que les informations initiales ont été communiquées.
La fonction est donc une boîte noire dissociant le traitement qui lui est interne, du dialogue avec l'utilisateur qui, lui, est externe à la fonction.

- cohérence: toute fonction, susceptible d'avoir une incidence sur la mémoire du système d'information, devrait garantir qu'au terme de son exécution elle a préservé l'intégrité de cette mémoire.

II.4.2. Décomposition de l'application

II.4.2.1. Fonctionnalités importantes:

Gestion d'un fichier
Réceptionne un nom de fichier, recherche dans le texte DSL qu'il contient la dénomination d'un schéma. Complète une table de correspondances entre noms de fichiers et noms de schémas.

Travail sur 2 schémas
Propose une paire de schémas à l'utilisateur, réceptionne celle choisie par ce dernier. Extrait des textes DSL correspondants à la sélection les listes des entités qu'ils contiennent.

Travail sur 2 entités
Construit une liste de correspondances précises entre les entités appartenant aux schémas sélectionnés dans la phase 2 à partir des correspondances entre attributs (si elles existent déjà)
Construit une liste de propositions de correspondances les entités appartenant aux schémas sélectionnés dans la phase 2 par transitivité ou par l'analyse des listes d'entités. Propose cette liste à l'utilisateur. Réceptionne les entités choisies par ce dernier. Extrait des textes DSL correspondants à la sélection, les listes des attributs qu'ils contiennent.

Travail sur 2 attributs
Construit une liste de correspondances entre les attributs appartenant aux entités sélectionnées dans la phase 3 à partir d'une analyse des listes de ces attributs. Propose cette liste à l'utilisateur. Réceptionne les attributs choisis par ce dernier.

Présentation de l'arbre des attributs (Tree)
Construit à partir de la liste des attributs d'une des entités sélectionnées dans la phase 3 une structure arborescente et la propose à l'utilisateur.

Présentation de la description d'un objet (Descr)
Réceptionne le choix de l'utilisateur concernant un objet appartenant à - ou étant un des schémas sélectionnés dans la phase 2. Extrait du texte DSL correspondant la description de cet objet et la propose à l'utilisateur.

Présentation du HELP (Help)
Propose à l'utilisateur un texte d'aide.

Ecriture d'une correspondance simple entre entités
Réceptionne une correspondance simple entre les entités sélectionnées

Ecriture d'une correspondance précise entre attributs sélectionnés
Réceptionne une correspondance précise entre les deux attributs sélectionnées.

Toutes ces phases sont interactives car elles contiennent au moins un message à destination ou en provenance de l'utilisateur.

II.4.2.2. Décomposition en fonctions:

fonction 1.1: extraction du nom du schéma dans le texte DSL
fonction 1.2: création de la table de correspondance fichier / schéma

fonction 2.1: recherche d'une paire de schémas de travail
fonction 2.2: constitution des listes des entités des schémas de travail

fonction 3.1: construction de correspondances précises entre entités à partir des correspondances entre attributs (si elles existent déjà)
fonction 3.2: recherche de propositions de correspondances entre entités par analyse par l'analyse des listes d'entités
fonction 3.3: recherche de propositions de correspondances entre entités transitivité
fonction 3.4: constitution des listes d'attributs des entités de travail

fonction 4.1: recherche de propositions de correspondances entre les attributs des deux entités sélectionnées.

fonction 5.1: Construction d'un arbre à partir d'une liste d'attributs.

fonction 6.1: Extraction d'une description d'objet d'un texte DSL.

fonction 7.1: Composition d'un texte d'aide.

fonction 8.1: Enregistrement d'une correpondance simple entre entités.

fonction 9.1: Enregistrement d'une correpondance précise entre attributs.

II.4.3. Schéma de la dynamique

"L'objectif de ce schéma est de fournir une visualisation des conditions de déclenchement, d'exécution et d'enchaînement des traitements en vue de caractériser les éléments du système d'information qui causent la production des messages résultats et les changements de la mémoire du système d'information." [Bodart,Pigneur 88].
Nous présentons le schéma de la dynamique relatif à notre application dans la figure 4.2. Dans un but de clarté, le schéma ne reprend que la découpe en fonctionnalités définie en II.4.2.1

=> un message interactif est un message en provenance ou à destination de l'utilisateur. Ils représentés en hachuré dans la figure 4.2.

=> un test interactif est un test basé sur l'analyse d'un message interactif. Une fonction réceptionne et évalue le message et envoie le résultat de son travail au test. Ce concept est représenté par le figure 4.1.

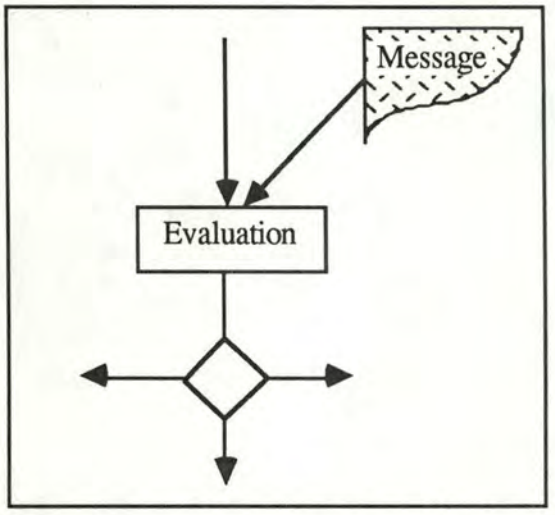


Figure 4.1: **Test interactif**

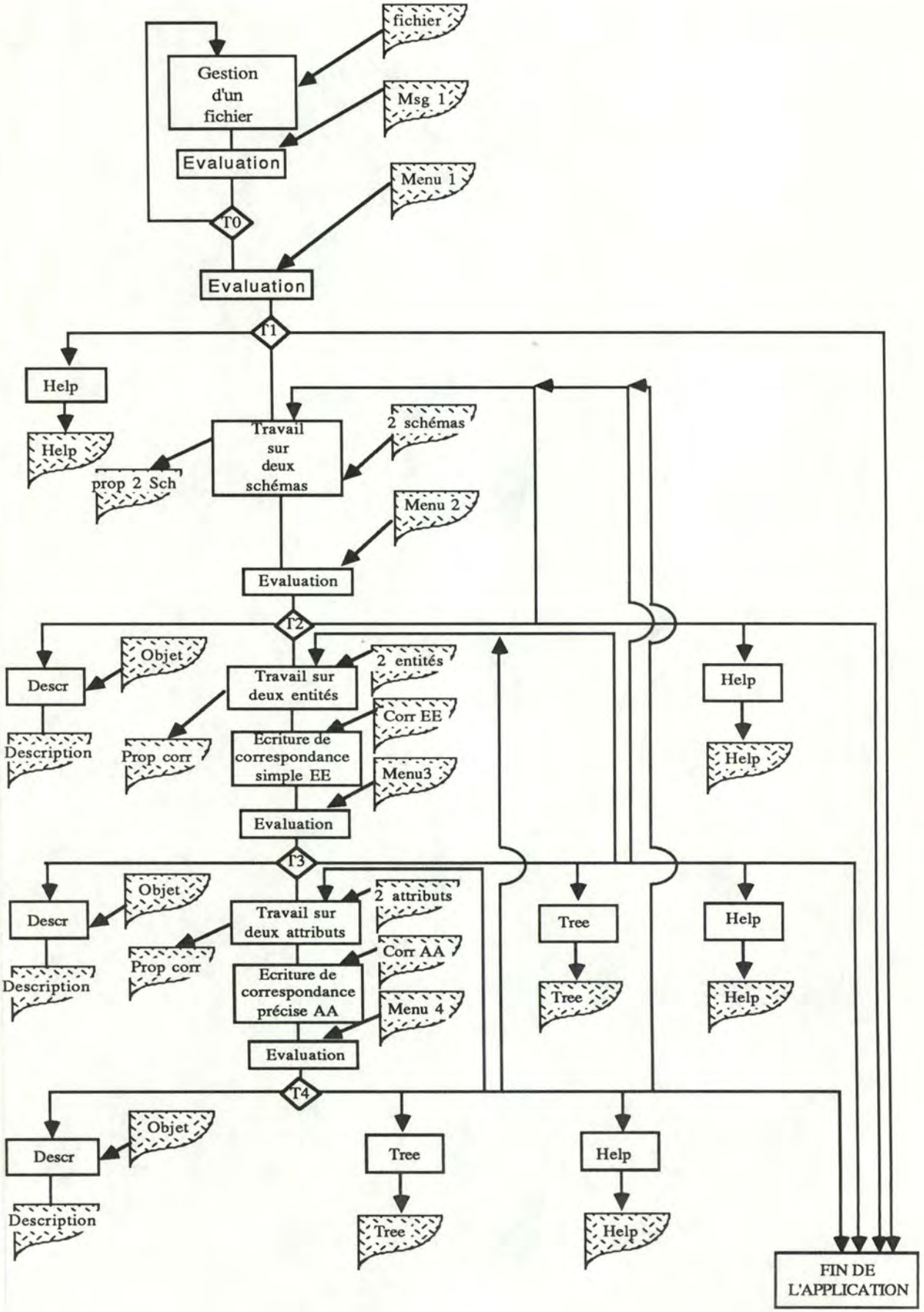
=> un objet inertactif est la visualisation d'un ou plusieurs messages interactifs. Cette définition autorise les objets interactifs emboîtés.

Concrètement, une fenêtre (1) mère est composée d'une ou plusieurs fenêtres filles et d'objets interactifs de bas niveau tels que les boutons, les message-box... La fenêtre fille peut elle même être mère et ainsi de suite.

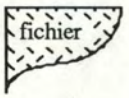
=> le premier message inclus dans un objet interactif est celui, parmis l'ensemble des messages interactifs qui composent l'objet, qui se trouve en amont des autres sur le schéma de la dynamique.

=> un objet interactif O1 en précède un autre O2, si le premier message inclus dans O1 précède dans le diagramme des flux le premier message inclus dans O2.

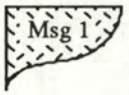
(1) Les concepts de Tiled Window, de fenêtre et de leurs composants sont des notions indispensables dans un environnement tel que celui fourni par MS-WINDOWS. Le lecteur pourra trouver toutes les explications nécessaires dans [Durant,Carlson,Yao].


fichier
Gestion d'un fichier
Msg 1
Evaluation
Menu 1
T0
Evaluation
T1
Help
Help
Travail sur deux schémas
2 schémas
prop 2 Sch
Menu 2
Evaluation
T2
Descr
Objet
Description
Travail sur deux entités
2 entités
Prop corr
Corr EE
Ecriture de correspondance simple EE
Menu3
Help
Help
Evaluation
T3
Descr
Objet
Description
Travail sur deux attributs
2 attributs
Prop corr
Corr AA
Ecriture de correspondance précise AA
Menu 4
Tree
Tree
Help
Help
Evaluation
T4
Descr
Objet
Description
Tree
Tree
Help
Help
FIN DE L'APPLICATION

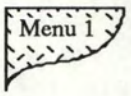
Signification des messages:



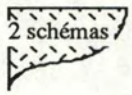
Nom d'un fichier contenant u n teste DSL



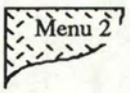
Next: nouveau fichier|| End: plus de fichier à entrer



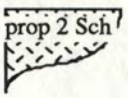
Help|| Selection de 2 schémas



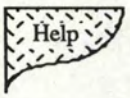
Deux noms de schémas



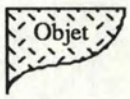
Help|| Description d'un schéma|| Sélection de 2 entités



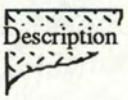
Proposition d'une paire de schémas à intégrer



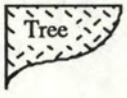
Texte d'aide à l'utilisateur



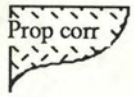
Désignation d'un objet



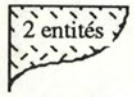
Texte de description d'un objet



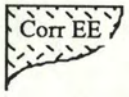
Arbre des attributs d'une entité



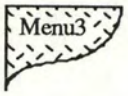
Proposition de correspondances entre 2 entités ou 2 attributs



Noms de 2 entités



Correspondance simple entre deux entités

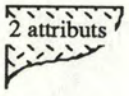


Help|| Description || Tree || Sélection de 2 attributs

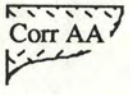
|| Sélection d'une paire d'entités

|| Sélection d'une paire de schéma

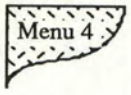
|| Sortie



Noms de 2 attributs



Correspondance précise entre 2 attributs



Help|| Description||Tree|| Sélection d'une paire d'attributs

|| Sélection d'une paire d'entités

|| Sélection d'une paire de schéma

|| Sortie

II.4.4. Analyse du schéma de la dynamique

Nous allons procéder à deux regroupements. D'une part, le regroupement des messages de manière à former des objets interactifs et d'autre part, le regroupement des fonctions, de manière à former des blocs dont chacun sera associé à un objet interactif.

II.4.4.1 Regroupement des messages

Posons le problème:
Un dialogue se déroule entre un programme et un utilisateur
Un message interactif provient soit du programme, soit de l'utilisateur et est à destination de l'autre participant au dialogue

Le bon sens et la facilité du travail de l'utilisateur donne comme critère de regroupement:
"Les messages interactifs ayant un fond sémantique commun doivent être regroupés".
Par exemple, la question "quelle est le n° de votre maison ?" devrait se trouver dans le même objet interactif que "quel est le nom de votre rue ?" car ces deux messages portent sur un concept commun: l'adresse.

Cependant, il faut restreindre ce critère de manière à ne pas aboutir à une impossibilté de production de certains objets. Nous définirons cette situation de la manière suivante:
Soit un regroupement de messages interactifs donnant un ensemble d'objets interactifs.
Chaque objet O possède un contenu sémantique égal à la somme du contenu des messages le composant. Nous décomposerons le contenu sémantique d'un objet en deux parties: celle déduite des messages provenant de l'utilisateur et celle déduite des messages provenant du programme.
Nous les nommerons respectivement O_{in} et O_{out}.

Soit O_n un objet interactif .
Soit O_{pre} un ensemble d'objets précédants O_n dans le schéma de la dynamique.

O_n est un rassemblement correct de messages si et seulement si :
c1- Le programme peut produire $O_{n\text{-}out}$:
Le contenu sémantique égal à la somme des contenus sémantiques de $O_{x\text{-}in}$, pour tout O_x appartenant à O_{pre} plus celui de $O_{n\text{-}in}$ suffit au programme pour produire $O_{n\text{-}out}$.

c2- L'utilisateur peut produire $O_{n\text{-}in}$:
Le contenu sémantique égal à la somme des contenus sémantiques de $O_{x\text{-}out}$, pour tout O_x

appartenant à Opre plus celui de On-out suffit à l'utilisateur pour produire On-in.

Il faut préciser que le concepteur définit strictement la sémantique nécessaire au programme pour qu'il produise une nouvelle information. L'utilisateur, lui, peut avoir besoin d'informations l'aidant à produire de nouvelles informations, c'est le cas du HELP par exemple. La condition c2 possède donc un sens plus extensible que la condition c1.

Les objets interactifs emboîtés sont traité de manière analogue.

L'objet interactif associé au bloc 1 est: **SelectFile**
Il est composé des messages interactifs fichier et Msg1

L'objet interactif associé au bloc 2 est: **SelectSchemas**
Il est composé des messages interactifs 2_Schémas et Prop_2_Schémas.

L'objet interactif associé au bloc 3 est: **SelectEntity**
Il est composé des messages interactifs: Prop_Corr_EE, 2_Entités.
et d'un objet interactif:

L'objet interactif associé au bloc 8 est: **Write_Corr_EE**
Il est composé du message interactif: Corr_EE.

L'objet interactif associé au bloc 4 est: **SelectAttribute**
Il est composé des messages interactifs: Prop_Corr_AA, 2_Attributs, Corr_AA.
et d'un objet interactif:

L'objet interactif associé au bloc 9 est: **Write_Corr_AA**
Il est composé du message interactif: Corr_AA.

L'objet interactif associé au bloc 5 est: **Tree**
Il est composé du message interactif: Tree.

L'objet interactif associé au bloc 6 est: **Description_Objet**
Il est composé des messages interactifs: Objet, Description.

L'objet interactif associé au bloc 7 est: **Help**
Il est composé du message interactif: Help

II.4.4.2 Regroupement des fonctions

=> un bloc est un groupe de fonctions dont au moins une reçoit ou produit un message interactif. Il peut également inclure des tests interactifs. Le bloc est associé à un objet interactif.
Celui-ci symbolise un ensemble d'informations à fournir à l'utilisateur ou à saisir en vue d'un traitement.

Nous proposons de rassembler en un bloc les fonctions suivantes:
- les fonctions recevant les messages interactifs constituant Oin
- les fonctions produisant les messages interactifs constituant Oout
- les fonctions ne recevant aucun message interactif comprises entre le premier message de l'objet interactif traité et le premier message inclus dans l'objet interactif en aval.

Les blocs doivent offrir un compromis entre deux exigences contradictoires.
- Ils peuvent s'exécuter dans un ordre le plus libre possible car le programme est interactif et, si le multitasking est offert, plusieurs blocs peuvent s'exécuter simultanément.
- Leur ordre d'exécution doit satisfaire le scénario défini dans la phase de spécification.

Le bloc 1 est composé des fonctions 1.1 et 1.2
Le bloc 2 est composé des fonctions 2.1 et 2.2
Le bloc 3 est composé des fonctions 3.1, 3.2, 3.3, 3.4 et 8.1
Le bloc 4 est composé des fonctions 4.1 et 9.1
Le bloc 5 est composé de la fonction 5.1
Le bloc 6 est composé de la fonction 6.1
Le bloc 7 est composé de la fonction 7.1

L'objet interactif associé à l'application est la **TILED_WINDOW**
Il est composé des objets interactifs définis ci-dessous, ainsi que des messages interactifs Menu1, Menu2, Menu3, Menu4.

L'architecture interne à chaque bloc sera définie séparément. Les relations entre blocs seronts définies dans le dialogue.

Capitre II.5: MODELISATION DU DIALOGUE

ERRATUM

Dans le chapitre I.5, le lecteur voudra bien lire les références aux schémas "Figure 5.x" au lieu de "Figure 4.x".

II.5. Modélisation du dialogue.

Une méthode classique pour modéliser l'aspect 'dialogue' d'une application est d'utiliser un diagramme de transition d'états comme celui proposé dans [Wasserman 85].
Dans ce diagramme, un noeud représente un état stable attendant un input de l'utilisateur; un arc représente une transition d'état basée sur un input particulier. Une action peut être associée à une transition pour représenter une opération qui doit être réalisée chaque fois qu'un arc particulier est traversé.
Le défaut des diagrammes d'états apparaît dans les systèmes où plusieurs fenêtres peuvent être affichées simultanément. En effet, dans ce cas, le nombre d'états que le système peut prendre devient vite énorme; et les diagrammes s'en trouvent compliqués à l'extrême. Le problême devient insurmontable si le système offre des possibilités de multitasking.

Pour palier à ces difficultés, nous proposerons un diagramme (cfr Figure 8.1) dans lequel les noeuds représentent les objets interactifs du système, plusieurs d'entre eux pouvant être actifs en même temps.
Chaque arc représente une action de l'utilisateur dans une fenêtre. L'action étant généralement le click via la souris ou le clavier sur un menu ou un bouton appartenant à la fenêtre origine de la flèche(cfr légende du schéma). Le résultat de cette action sera l'activation de la fenêtre destination de la flèche. La fenêtre destination sera donc fille de la fenêtre origine.
Nous pouvons remarquer qu'il existe des fenêtres (1) (Tree,Descr,Help) ne possèdant pas de flèches de retour vers la Tiled Window. Ceci pour marquer le fait que lorsque l'une de ces fenêtres est activée, d'autres fenêtres et dialogbox peuvent l'être également alors que la première est toujours visible. Donc, fermer ou non une fenêtre n'est d'aucune importance (sauf restriction explicite du programmeur, ex: les dialogbox) pour le reste de l'application et ce, en vertu du multitasking offert par MS-Windows. On spécifie quelles sont les activités bloquantes en plaçant des flèches de retour sur les objets interactifs concernés. Ces flêches de retour signalent la cloture d'une activité bloquante.

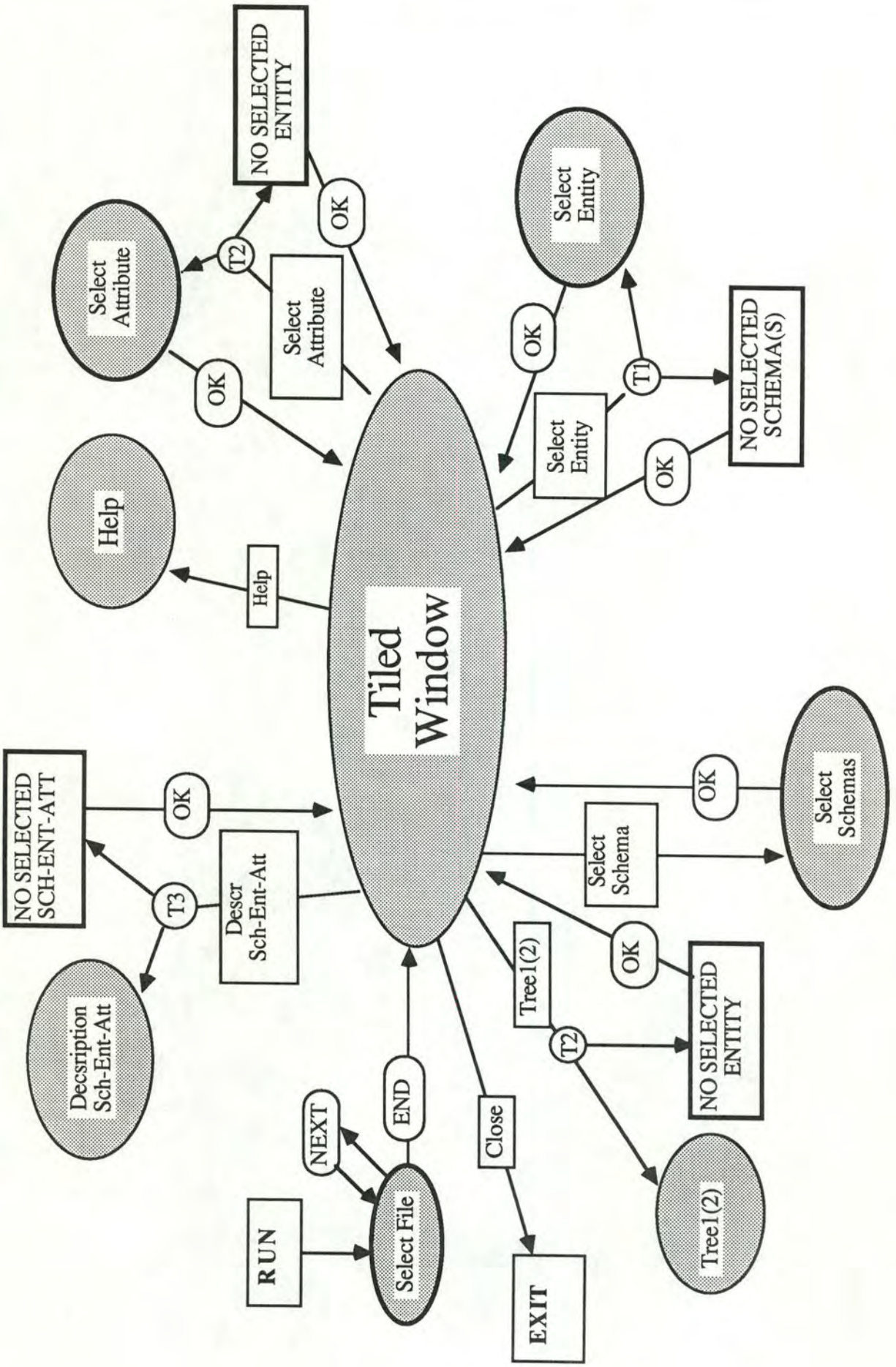
(1) En règle générale, l'implémentation d'un objet interactif est une fenêtre; celle d'un test interactif est un menu.

Il peut exister des contraintes d'ordre dans l'activation des objets interactifs; pour introduire ces relations d'ordre dans le schéma nous avons placé des tests sur les arcs. Ces test sont des OU exclusif. En cas de détection d'une erreur, l'effet sera l'apparition d'une messagebox la signalant à l'utilisateur. ou par par la désactivation ('mise en grisé') de certaines options menu lorsque ces conditions ne sont pas satisfaites. L'exemple classique dans l'application est qu'on ne peut travailler sur une entité avant d'avoir sélectionner un schéma.

Les rectangles nommés Run et Exit symbolisent respectivement l'entrée et la sortie du programme.

Le cas des objets interactifs emboîtés peut être représenté par des diagrammes du dialogue emboîtés, comme il est illustré par les figures 5.2, 5.3, 5.4. Un objet O interactif est représenté dans un diagramme de niveau N(≥0, le niveau 0 est celui où intervient la Tiled Window). Cet objet peut contenir d'autres objets emboîtés, soient O1 et 02. O1 et O2 ne sont pas visualisés dans le diagramme de niveau N mais bien à un niveau N-1. Dans ce sous-diagramme(cfr Figures 8.3 et 8.4), c'est O qui fera office de Tiled Window. Ceci est illustré dans la figure 5.4.

Page suivante: Figure 4.1: **Modélisation du dialogue**


Tiled Window
RUN
Select File
NEXT
END
Close
EXIT
Help
Help
Select Attribute
OK
T2
NO SELECTED ENTITY
Select Attribute
OK
Select Entity
OK
T1
NO SELECTED SCHEMA(S)
Select Entity
OK
NO SELECTED SCH-ENT-ATT
OK
T3
Descr Sch-Ent-Att
Decsription Sch-Ent-Att
Select Schema
OK
Select Schemas
Tree1(2)
T2
OK
NO SELECTED ENTITY
Tree1(2)

Légende

T1: Un schéma a-t-il été sélectionné ?

T2: Une entité a-t-elle été sélectionnée ?

T3: Un objet (Sch,Ent,Att) a-t-il été sélectionné ?

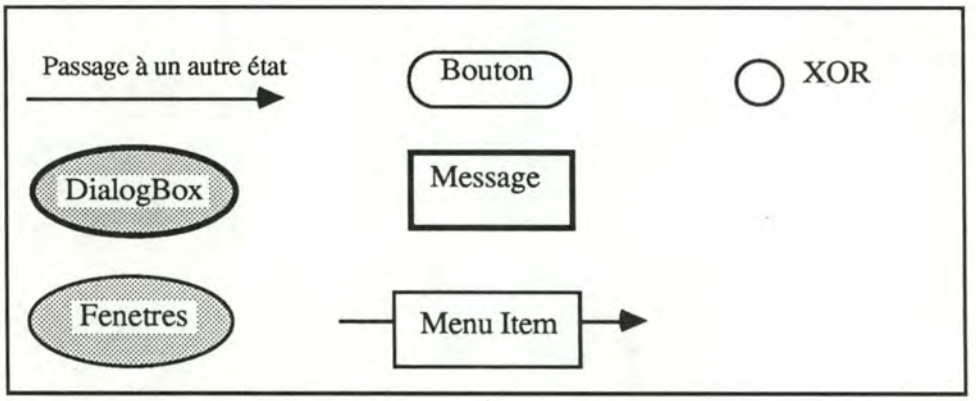


Figure 5.2:**Légende**

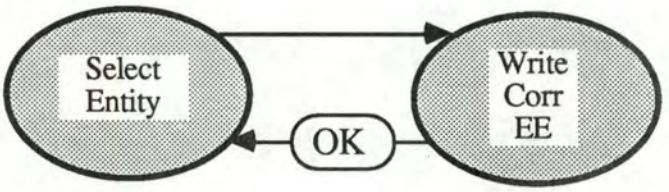


Figure 4.3: **Sous-dialogue de l'objet interactif "Select Entity"**

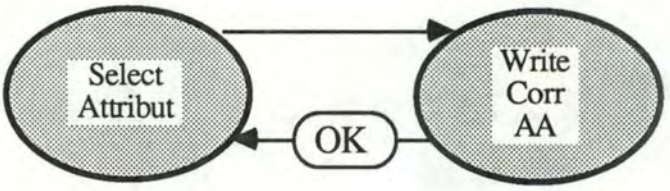


Figure 4.4: **Sous-dialogue de l'objet interactif "Select Entity"**

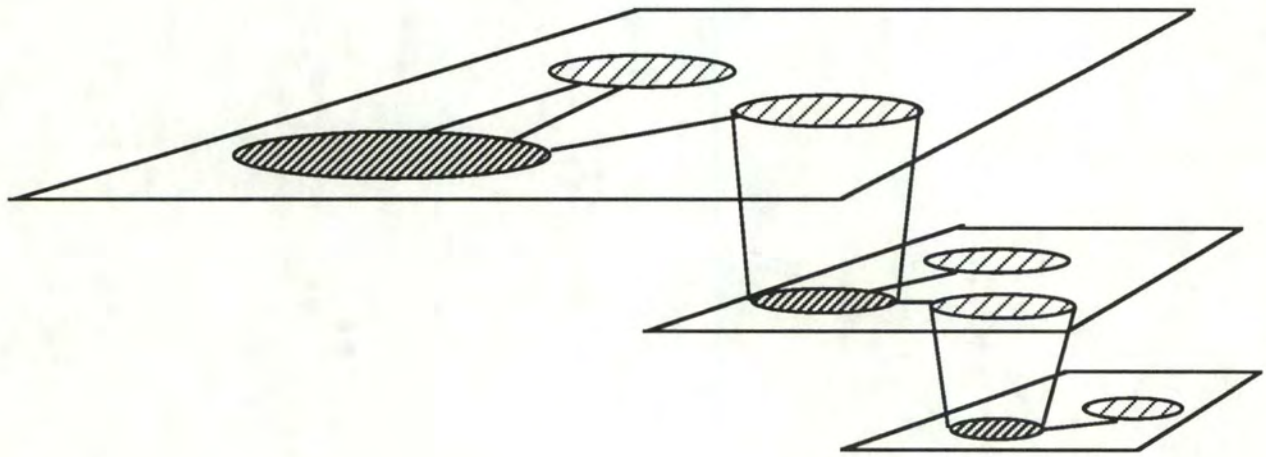


Figure 4.5: **Dialogues emboîtés**

Les ovales hachurés en foncé symbolisent les fenêtres principales dans leur (sous) dialogue.

Chapitre II.6: ARCHITECTURE LOGIQUE

II.6.1. Données manipulées

II.6.2. But des fonctions

II.6.3. Architecture interne à un bloc

II.6. Architecture logique

Nous allons procéder dans ce chapitre à la spécification des fonctions. Ces fonctions sont dérivées directement de celles utilisée dans le chapitre II.4. Nous donnerons pour chacune d'elles quelques lignes exprimant son utilité. Le lecteur pourra trouver des spécifications précises sous forme pré/post-conditions en annexe A1. Nous terminerons par l'architecture interne de chacun des blocs définis dans le chapitre II.4.

L'architecture présentée s'articule autour de la relation "utilise" dont nous rappelons la définition:

=> Un module A utilise un module B si l'exécution correcte de A dépend de l'existence d'une version correcte de B. Une version correcte de A sous-entend que A remplisse ses spécifications.

Nous allons tenter de satisfaire les buts d'une "bonne architecture" au sens où on nous l'a enseigné. C'est à dire:
Spécifications correctes des modules
Réutilisation d'un maximun de modules
Minimun de couplage entre les modules
Information hiding
Nous nous limiterons à un sous-système aidant à la détection des correspondances entre entités ainsi qu'entre les attributs de ces dernières.

II.6.1 Données manipulées par le programme:

- Une table de correspondance entre les noms des fichiers DSL et le nom du schéma qu'ils contiennent.
- Les noms des deux schémas sélectionnés.
- Les noms des deux entités sélectionnées.
- Les noms des deux attributs sélectionnés.
- Les deux listes contenant les noms des entités appartenant aux deux schémas sélectionnés.
- Les deux listes contenant les noms des attributs appartenant aux deux entités sélectionnés et ce, sous une forme arborescente, c'est-à-dire en précisant si un attribut est le frère ou le fils d'un autre.
- La liste des correspondances déja détectées entre les deux schémas sélectionnés.

II.6.2 But des fonctions

Transit:
Propose de nouvelles correspondances déduites par transitivité des correspondances déjà détectées.

Denom-Id:
Détecte les dénominations identiques se trouvant dans deux listes d'objets (attributs ou entités).

Quasi-Id:
Détecte les dénominations quasi-identiques (abréviations) se trouvant dans deux listes d'objets (attributs ou entités).

Syn:
Détecte les synonymes éventuels se trouvant dans deux listes d'objets (attributs ou entités) en se basant sur un fichier "fsyn" contenant des listes de synonymes fréquemment utilisés dans le monde modélisé par les schémas DSL.

ReadSdc:
Extrait du fichier SDC les correspondances concernant les deux schémas sélectionnés.

ReadDsl:
Extrait du fichier DSL des informations concernant un objet sélectionné (fichier, schéma, entité, attribut).

ListEntity:
Fournit les deux listes d'entités appartenant aux deux schémas sélectionnés.

ListAtt:
Fournit les deux listes d'attributs appartenant aux deux entités sélectionnées.

PropCorr:
Construit un ensemble de proposions de correspondances entre deux listes d'objets (entités ou attributs) appartenant aux deux schémas sélectionnés.

NewCorr:
Gère les traitements nécessaires à la saisie d'une correspondance.

EE_Simple:
Gère la détection d'une liste de correspondances simples entre les entités de deux schémas sélectionnés ainsi que la sélection d'une paire d'entité de travail.

Precise_Att:
Gère la détection d'une liste de correspondances précises entre les attributs de deux entités sélectionnées ainsi que la sélection d'une paire d'attributs de travail.

AlgoCorr:
Calcule la correspondance précise existant entre deux entités à partir des correspondances précises détectées entre leurs attributs.

Precise_EE:
Fournitune liste de correspondances précises entre les entités de deux schémas sélectionnés.

ListFile:
Gère la saisie correcte des noms des fichiers DSL sur lesquels le programme va travailler.

Sch/File:
Construit une table de correspondance entre les noms des fichier DSL et les noms des schémas contenus dans ces fichiers (un par fichier).

SelectSchema:
Gère la saisie d'une paire de schémas de travail.

PropSch:
Propose un ordre d'intégration des schémas.

WriteSdc:
Met à jour la partie du fichier SDC concernant les deux schémas sur lesquels on vient de travailler.

ControlSel...:

Contrôle de la validité d'un choix de l'utilisateur

PrepData...:

Formate les données en vue de leur affichage

II.6.3 Architecture interne à un bloc

Les figures de 6.1 à 6.8 représentent l'architecture interne à chaque bloc.
Les traits symbolisent des relations "utilise".
Lorsque les noms de modules sont identiques dans deux arbres, il est bien entendu qu'il sagit d'un seul et même module.

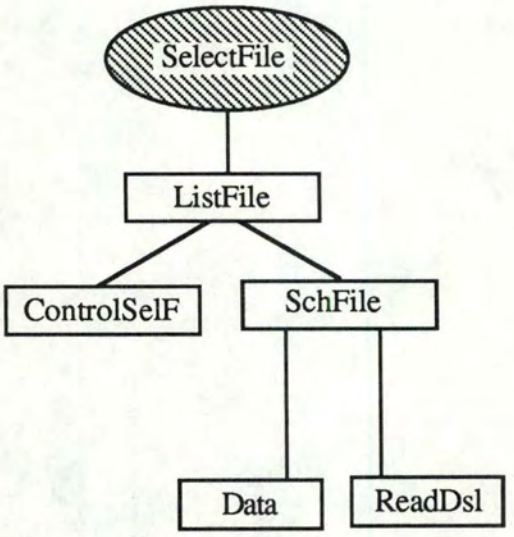


Figure 6.1

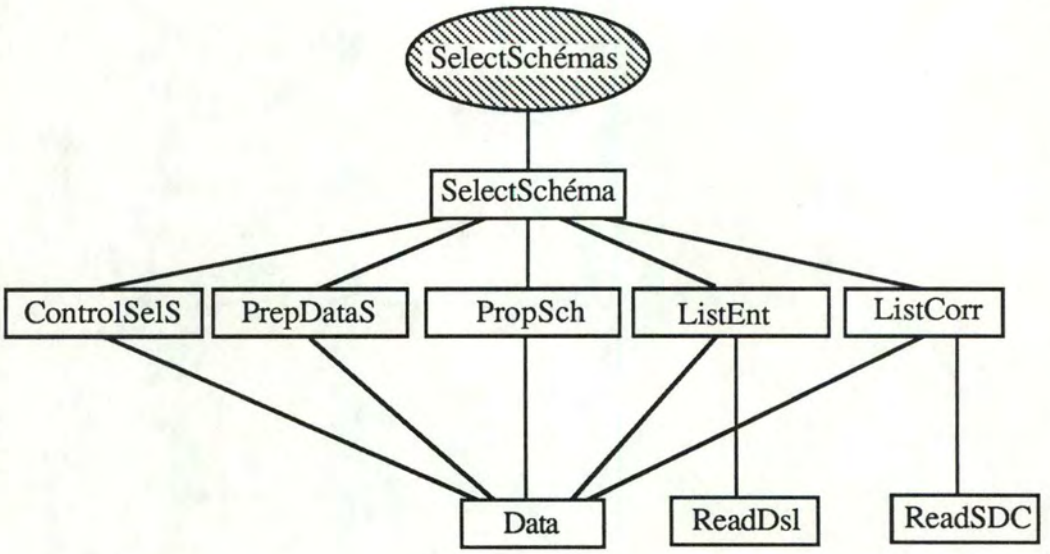


Figure 6.2

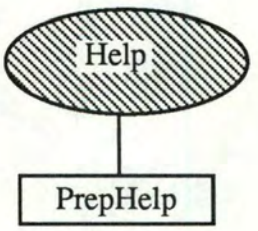


Figure 6.3

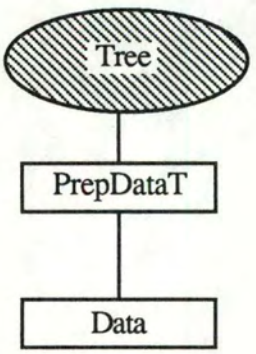


Figure 6.4

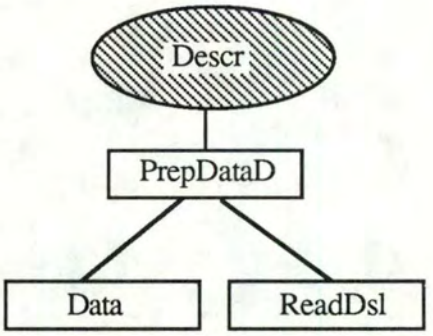


Figure 6.5

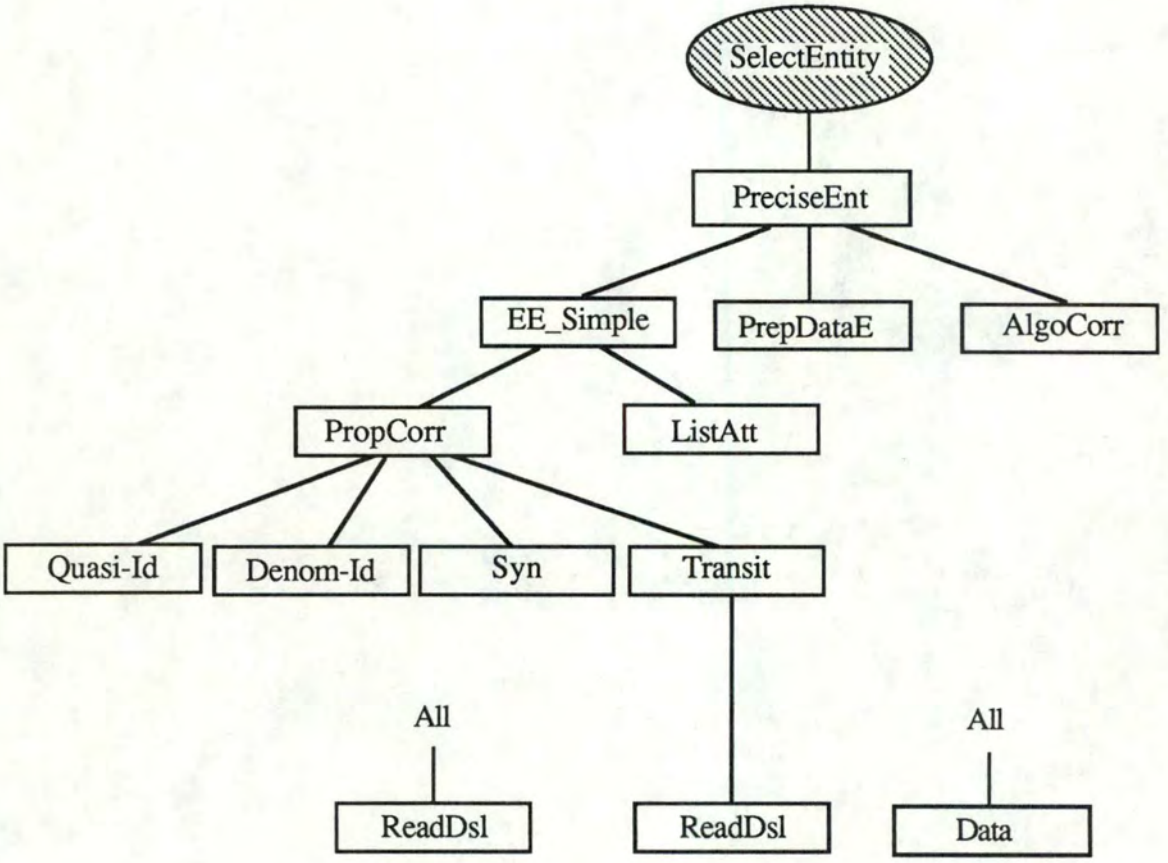


Figure 6.6

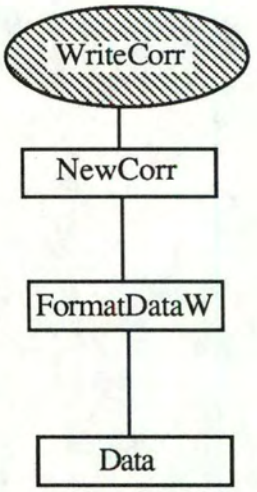


Figure 6.7

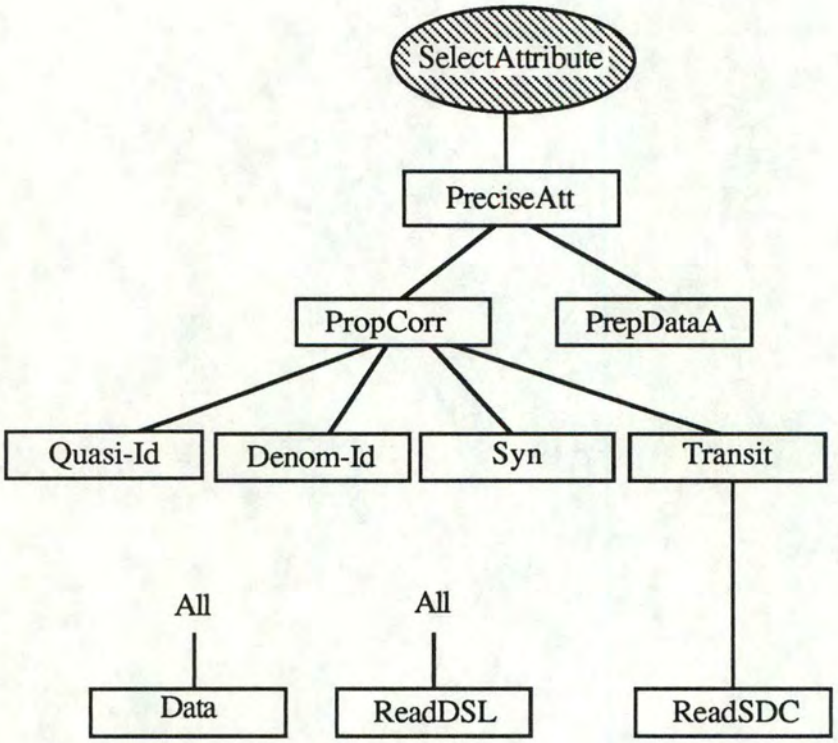


Figure 6.8

Conclusion générale

Dans ce travail, nous avons vu que le processus d'intégration pouvait être avantageusement conçu comme la succession de deux phases: la recherche de correspondances et l'intégration proprement dite.
C'est certainement la première de celles-ci qui constitue la partie la plus intéressante et la plus difficile du problème, car la sémantique des objets entre en jeu. Il est donc impossible d'automatiser complètement cette phase.

Une première chose indispensable pour faciliter le travail de l'utilisateur est de disposer d'un jeux de types de correspondances judicieusement choisi et bien adapté au modèle de données utilisé. C'est ce que nous avons tenté d'obtenir en partant d'une classification générale des correspondances que nous avons transposée dans le modèle ERA. Contrairement à ce que nous avons trouvé dans la littérature, nous avons voulu donner une définition très précise des correspondances. Cela nous a permis de mettre en lumière certaines déductions possibles sur des ensembles de correspondances.
Ce type de démarche devrait être privilégié, car en plus de diminuer le travail de l'utilisateur, il réduit le risque d'incohérence dans l'ensemble des correspondances entrées. Cependant, ce risque reste toujours présent, c'est pourquoi il est nécessaire d'opérer des contrôles de cohérence. Ce traitement est aussi facilité par le fait de disposer de correspondances précises, et mériterait peut-être d'être étudié plus que nous ne l'avons fait.

A la lecture des définitions données dans le chapitre I.4, on se rend bien compte que la détection d'une correspondance n'est pas un problème simple. C'est pourquoi, en plus des déductions possibles, il est souhaitable de donner à l'utilisateur un maximum d'indices ou de propositions qu'il pourra choisir de valider à sa convenance. Nous avons proposé quelques outils dans ce sens, mais il ne sont absolument pas exhaustifs.
Un cas intéressant est celui de l'outil chargé de la détection des synonymes. Il travaille sur un fichier contenant une liste des synonymes les plus souvent utilisés dans le domaine d'application des schémas DSL à intégrer. Son originalité est qu'il peut être complèté par des utilisations succesives du programme. Par exemple, si l'utilisateur déclare que deux objets sont en correspondance, il paraît logique de penser que leurs dénominations sont synonymes. Et donc, le programme peut augmenter le fichier de cette paire de dénominations, s'il ne la possède pas encore.

Au fur et à mesure de son utilisation, le programme devrait devenir de plus en plus performant dans le sens où il proposerait de plus en plus de correspondances à l'utilisateur avec un déchet de plus en plus faible. Cette observation montre qu'il serait intéressant d'introduire des concepts de systèmes experts dans la recherche de correspondances. Nous n'avons pas étudié cette approche, mais elle nous semble très prometteuse.

Les considérations qui précèdent montrent bien toute l'importance que revêt le dialogue avec l'utilisateur, c'est pourquoi nous avons insisté particulièrement sur ce point dans la description de notre implémentation. Actuellement, il existe peu de travaux traitant de la conception de programmes interactifs, mais il semblerait qu'il existe un consensus pour dire que dans ce domaine, les problèmes proviennent de deux grandes causes. La première est de construire des architectures qui ne séparent pas clairement la 'couche présentation' des fonctionnalités de l'application. La seconde est de sous-estimer l'aspect dynamique des applications interactives dans lesquelles c'est l'utilisateur, qui dans une certaine mesure, décide de l'enchaînement des fonctionnalités. Ce problème constitue en lui-même un immense champ de recherches. Pour notre part, nous avons choisi de livrer quelques réflexions qui devraient faciliter l'affrontement de ces difficultés.

BIBLIOGRAPHIE

[Batini, Lenzerini 82]

C. Batini, M. Lenzerini
A methodology for data schema integration in the entity-relationship model
Universita di Roma, 1982

[Batini,Lenzerini,Navathe 86]

C. Batini, M. Lenzerini, S. Navathe
A comparative analysis of methodologies for database schema integration
ACM Computing Surveys, dec 1986

[Belfar 84]

Khedidja Belfar
Méthode d'intégration de schémas et application aux schémas
exprimés dans un modèle de type E-R-A
Thèse de doctorat, 1984
Université Pierre et Marie Curie (PARIS VI)

[Bergland 87]

G. D. Bergland
A guided tour of program design methodologies
IEEE, oct 1981

[Biskup,Convent 86]

Joachim Biskup, Bernhard Convent
A formal view integration method
ACM 1986

[Bodart, Pigneur 83]

François Bodart, Yves Pigneur
Conception assistée des applications informatiques
Vol.1 Etude d'opportunité et analyse conceptuelle
Editions Masson, 1983
Une nouvelle version de cet ouvrage est prévue pour
le courant de l'année 1988

[Convent 86]

Bernhard Convent
Unsolvable problems related to the view integration approach
Universitat Dortmund, 1986

[Coutaz 87]

Joëlle Coutaz
Méthodes et outils pour l'implémentation des interfaces modernes
Cours INRIA 1897

[D'Attri, Sacca 84]

A. D'Attri, D. Sacca
Equivalence and mapping of database schemes
Proceedings of the 10th VLDB conference, 1984

[Dayal,Hwang 84]

Umeshwar. Dayal, Hai-Yann Hwang
View definition and generalization for database integration
in a multidatabase system
IEEE, nov 1984

[Durant, Carlson, Yao]

David Durant, Geta Carrlson, Paul Yao
Programmer's guide to Windows
Sybex

[Elmasri, Wiederhold 79]

Ramez Elmasri, Gio Wiederhold
Data model integration usign the structural model
SIGMOD 79

[Grison 87]

Thierry Grison
Etat de l'art en intégration des vues d'une base de données
Université de Bourgogne
Rapport de recherche n°8703-mai 1987

[Jajodia,Springsteel 83]

Sushil Jajodia, Frederick Springsteel

The problem of equivalence for entity-relationship diagrams

IEEE, sept 1983

[Mannino,Effelsberg 84]

Michael Mannino, Wolfgang Effelsberg

A methodology for global schema design

University of Florida 1984

[Navathe, Elmasri, Larson 86]

Shamkant Navathe, Ramez Elmasri, James Larson

Integrated user views in database design

Computer, vol 19, 1986

[Parent 87]

Christine Parent

L'approche ERC: Un modèle et une algèbre de type Entité-Relation

Thèse de doctorat

Université P. et M. Curie (Paris VI) 1987

[Spaccapietra, Demo, Parent 83]

S. Spaccapietra, B. Demo, C. Parent

A system for integrating existing heteregeneous distributed databases and applications programs

Université P. et M. Curie 1983

[Wasserman 85]

Anthony Wasserman

Extending state transaction diagrams for the specification of human-computer interaction

IEEE Transactions on software engineering, aug 1985

ANNEXES

Annexe 1: Spécifications des modules

Annexe 2: Description formelle des fenêtres

Annexe 3: Texte du programme

Annexe 4: Fichier des synonymes

SPECIFICATIONS DES MODULES DU SOUS-SYSTEME UTILE

Fichiers utilisés:

f_descr-i:

Ce fichier contient la description sous forme DSL du ieme schéma appartenant à l'ensemble des n schémas à intégrer.

fcorr:

Ce fichier contient la description sous forme SDC [cfr chapitre I.5] des correspondances détectées entre des objets appartenant aux n schémas à intégrer
Pour chaque correspondance se trouvant dans ce fichier, on a les informations suivantes:

* le texte SDC décrivant la correspondance
* un indicateur signalant si la correspondance est donnée sous sa forme précise ou simple.
* un idicateur signalant si la correspondance a été détectée automatiquement ou si elle a été donnée par l'utilisateur

Une correspondance peut se trouver au plus une fois dans fcorr.

fsyn:

Ce fichier contient les listes de synonymes sous le format suivant:

synonyme 1.1;synonyme 1.2;...;synonyme 1.N / 1
synonyme 2.1;synonyme 2.2;...;synonyme 2.N / 2
. . .
. . .
. . .
synonyme M.1;synonyme M.2;...;synonyme M.N / M

Programme

ENTREES:
* n fichiers f_descr_i

PRE:
* n fichiers en entrée contiennent les textes descriptifs des n schémas entre lesquels nous cherchons les correspondances.

SORTIES:
* le fichier texte fcorr contenant les correspondances détectées entre les entités des n schémas et les correspondances entre les attributs de ces entités.

POST:
* Le fichier résultat fcorr contient les correspondances précises entre les entités et les attributs appartenant aux n schémas spécifiés.
* La cohérence de cet ensemble de correspondances doit être établie par le module "Cohér" que nous n'implémenterons pas.

EFFET:
* Le fichier fsyn est augmenté des paires de noms correspondant aux correspondances détectées.

PreciseEnt

ENTREES:
* les noms de 2 schémas

PRE:
* les schémas sont différents et appartiennent à l'ensemble des n schémas à intégrer

SORTIE:
* un booléen

EFFET:
* le fichier texte fcorr contenant la liste des correspondances est augmenté des correspondances précises détectées entre les entités des 2 schémas donnés en entrée.
* le fichier fyn est augmenté, si nécessaire, et à la place adéquate, des désignations des entités concernées par les correspondances détectées.

POST:
* le booléen en sortie est vrai si au moins une correspondance précise a été détectée entre les 2 schémas traités, il est faux sinon.
* les correspondances concernant les 2 schémas identifiés en entrée sont toutes sous forme précise.
* si le fichier contient une correspondance entre deux entités appartenant aux deux schémas spécifiés en entrée, alors, il contient également les correspondances entre les attributs de ces entités. L'inverse est vrai également.

AlgoCorr

ENTREES:
* deux noms d'entités
* liste des correspondances entre attributs des entités

PRE:
* les entités appartiennent à des schémas différents

SORTIES:
* la correspondance précise entre les entités

Coher

ENTREES:
* 2 noms de schémas

PRE:
* les deux schémas nommés sont les derniers entre lesquels au moins une correspondance à été trouvée.

SORTIE:
* un booléen

EFFET:
* on a supprimé du fichier fcorr les correspondances qui introduisaient l'incohérence (choix effectué par l'utilisateur), ainsi que celles déduites par transitivité des correspondances sus-citées. Bien sur, si on supprime une correspondance entre 2 entités, on doit également supprimer les correspondances entre les attributs de ces entités.

POST:
* le fichier est cohérent
* il contient au plus toutes les correspondances du fichier initial et aucune autre n'appartenant pas à cet ensemble.
* le booléen en sortie est vrai si l'ensemble des correspondances donné en entrée est cohérent, il est faux sinon.

La notion d'incohérence dans un ensemble de correspondances a été définie dans le chapitre I.7.

PreciseAtt

ENTREES:
* désignations de 2 objets
* type du premier objet (entité ou relation)
* type du second objet (entité ou relation)

PRE:
* les 2 objets appartiennent à 2 schémas différents
*une correspondance sous forme simple a déja été détectée entre ces deux objets

SORTIE:
* un booléen

EFFET:
* le fichier fcorr a été complété par les correspondances détectées entre les attributs des 2 objets spécifiés en entrée

POST:

* le booléen en sortie est vrai si au moins une correspondance précise a été détectée entre les 2 schémas traités, il est faux sinon.

EE simple

ENTREES:

désignation de 2 schémas

PRE:

* l'ensemble des correspondances se trouvant dans fcorr est cohérent

SORTIE:

* un booléen

EFFET:

* le fichier fcorr est complété par de nouvelles correspondances simples détectées entre les entités des 2 schémas identifiés en entrée

POST:

* le fichier fcorr contient au moins toutes les correspondances qu'il possédait en entrée.
* le booléen est vrai si au moins une correspondance a été détectée entre les entités des 2 schémas traités, il est faux sinon.

Propcorr

ENTREES:

* Deux listes de noms d'objets entre lesquels on désire des propositions de correspondances.
* Deux types d'objets
* Deux noms de schémas

PRE:

* Les objets de chaque liste appartiennent tous respectivement à un des schémas spécifiés
* Les objets de chaque liste sont tous du type spécifié respectivement par un des deux types donnés en entrée.

* Les schémas spécifiés sont différents

SORTIES:
* Une liste de correspondances simples.

POST:
* Les deux objets repris dans chaque correspondance appartiennent respectivement à une des listes données en entrée.

Syn

ENTREES:
* 2 listes de noms d'objets

PRE:
* les objets contenus dans une même liste appartiennent à un même schéma
* Les listes contiennent des objets de deux schémas différents

SORTIE:
* une liste éventuellement vide de couples de noms

POST:
* les objets repris dans un des couples donnés par ce module ont des désignations synonymes c'est-à-dire qu'ils sont décrits comme tels dans le fichier fsyn.
* Les objets repris dans un couple n'appartiennent pas à la même liste en entrée

Denom-Id

ENTREES:
* 2 listes de noms d'objets

PRE:
* les objets contenus dans une même liste sont de même type et appartiennent à un même schéma
* chaque liste spécifiée appartient à unschéma différent

SORTIE:

* une liste éventuellemnt vide de couples de noms

POST:

* les objets concernés par un couple donné par ce module ont des désignations identiques.
* Les objets repris dans un couple n'appartiennent pas à la même liste en entrée.

Quasi-Id

ENTREES:

* 2 listes de noms d'objets

PRE:

* les objets contenus dans une même liste sont de même type et appartiennent à un même schéma
* chaque liste spécifiée appartient à un schémas différent.

SORTIE:

* une liste éventuellemnt vide de couples de noms
* Les objets repris dans un couple n'appartiennent pas à la même liste en entrée.

POST:

* les objets concernés par un couple donné par ce module ont des désignations quasi-identiques.

ReadDsl

interfaces:

* *descr_obj*
  IN: Type d'objet cfr définition SDC
  Désignation d'objet cfr définition SDC

  OUT: Description de l'objet donné en IN

* *listent*
  IN: nom de schéma

OUT: Liste des noms de toutes les entités appartenant au schéma spécifié

* *listatt*

IN: nom de schéma
nom d'entité

OUT: Liste des noms de touts les attributs appartenant à l'entité spécifiée

* *listsch*

IN: une liste de noms de fichiers contenant des schémas sous le formalisme DSL (un schéma par fichier)

OUT: Création d'une table de correspondance entre le nom du fichier et le nom du schéma qu'il contient. Cette fonction permet à l'utilisateur de s'exprimer en termes de schémas .

ReadSdc

Les procédures suivantes interrogent ou agissent sur le fichier des correspondances déja trouvées.

* *initsdc*

IN: Liste des noms de schémas sur lesquels on désire travailler

OUT: Initialisation du fichier fcorr avec les éléments MS et CS entre chaque paire de schémas

* *clc*

IN: Désignation de deux schémas

OUT: Liste des correspondannces déja détectées entre les objets appartenant aux schémas donné.

**pos_cor*

IN: Désignation de schéma
Désignation d'objet (entité ou relation) appartenant au schéma ci-dessus
Désignation d'un deuxième schéma

OUT: Expression, sous forme SDC d'une correspondance, si elle existe, entre l'objet spécifié en

IN et un autre objet appartenant au deuxième schéma spécifié en IN. S'il n'existe aucune correspondance satisfaisant ces paramètres, il renvoie la valeur vide.

cor_att

IN: Type d'objet: entité ou relation
Désignation d'objet cfr SDC

OUT: Liste des correspondances entre les att

* *circuit*

IN: une correspondance sous forme SDC

OUT: la liste des correspondances formant le circuit auquel appartient la correspondance donnée en IN.

* *del_cor*

IN: une correspondance sous forme SDC

OUT:

EFFET: enlève du fichier des correspondances déja détectées, la correspondance spécifiée en IN ainsi que toutes celles qui ont été déduites par transitivité à partir cette dernière en une ou plusieurs "générations".

* *wlc*

IN: liste de correspondances détectées entre deux schémas

OUT:

EFFET: mise à jour du fichier fcorr avec les correspondances données en entrée

Transit

ENTREES:

* Deux noms de schémas

PRE:

* les noms sont ceux des deux deerniers schémas entre lesquels une correspondance à été détectée.
* l'ensemble de correspondances déja trouvé est cohérent.

SORTIES:

POST:

*Le fichier fcorr est completé par les correspondances trouvées par transitivité à partir de celles se trouvant dans le fichier en entrée.
Ce nouvel ensemble de correspondances est cohérent.

Selectschema

ENTREES:

* Liste des noms de schémas sur lesquels on travaille

SORTIE:

* Deux schémas sélectionnés

POST:

*les schémas sélectionnés sont différents

SchFile

ENTREES:

* Liste des noms des fichiers contenant les descriptions des schémas sur lesquels on désire travailler

PRE:

* Un fichier contient un seul schéma
* Les textes des schémas sont bien-formés

SORTIES:

* Table de correspondance entre les noms des fichiers en entrée et les noms des schémas correspondants.

ListFile:

EFFET:
*constitue la table de correspondance entre noms de fichiers et noms de schémas DSL

PropSch

ENTREES:
* Liste des noms de schémas

SORTIES:
* Proposition d'une paire de schémas sur lesquels travailler

POST:
* L'utilisateur n'a encore introduit aucune correspondance entre les deux schémas proposés

NewCorr:

EFFET:
*La liste des correspondces entre tous les objets de schémas de travail sera chargée en mémoire.

ListEntity

ENTREES:
* Nom d'un schéma

SORTIES
* Liste des entités du schéma en entrée

ListAtt

ENTREES:
* nom d'un schéma
* nom d'une entité

PRE:
* l'entité appartient au schéma spécifié

SORTIES:
* Liste des attributs de l'entité en entrée

ControlSelF

ENTREES:
* string

SORTIES:
* booléen

POST:
* Le booléen est vrai si le nom donné en entrée est celui d'un fichier appartenant à la directory.

ControlSelS

ENTREES:
* string

SORTIES:
* booléen

POST:
* Le booléen est vrai si le nom du schéma rentré est différent de l'autre schéma de travail.

EFFET:
* si le test est positif, inscrit le string dans la variable représentant un des deux schémas de travail

PrepDataS

ENTREES:

SORTIES:
* liste de noms

POST:
* la liste de nom correspond au format nécessité par l'interface d'affichage
* la liste représente les n schémas à intégrer

PrepDataE

ENTREES:

SORTIES:
* 2 listes de noms

POST:
* les listes de noms correspondent au format nécessité par l'interface d'affichage
* les listes représentent les entités des deux schémas de travail

PrepDataA

ENTREES:

SORTIES:
* 2 listes de noms

POST:
* les listes de noms correspondent au format nécessité par l'interface d'affichage
* les listse représentent les attributs des entités de travail

PrepDataT

ENTREES:
* un nom

PRE:
* le nom donné en entrée est celui d'une entité de travail

SORTIES:
* liste de noms

POST:
* la liste de nom correspond au format nécessité par l'interface d'affichage (forme d'arbre)
* la liste représente les attributs de l'entité de travail donnée en entrée

PrepDataD

ENTREES:
* un nom
* un type

PRE:
* il existe un ojet du nom et dut ype donnés en entrée dans un des schémas de travail

SORTIES:
* texte

POST:
* le texte correspond au format nécessité par l'interface d'affichage
* le texte est la description de l'objet dont le nom est donné en entrée.

NewCorr:

ENTREES:

* deux noms de schémas
* deux types d'objet
* un type de correspondance

PRE:
* Les objets appartiennent aux schémas spécifiés

SORTIES:
* une correspondance entre les deux objets de travail de type donné en entrée.

FormatDataW:

ENTREES:
* type de correspondance
* type d'objet

EFFET:
* met à jour la liste des correspondances entre les deux schémas de travail

ReadData:

Ce module sert à accéder aux données manipulées par le programme. Ces données ont été citées en II.6.1.
Nous accéderons aux données par l'intermédiaire des fonctions spécifiées ci-dessous:

a] les noms des deux schémas de travail:

lire les noms
changer les noms

b] les noms des deux entités de travail:

lire les noms
changer les noms

c] les noms des deux attributs de travail:

lire les noms
changer les noms

d] les deux listes contenant les noms des entités des schémas de travail:

Pour chaque liste:

se positionner en début de liste
lire le nom courant
passer à l'élément suivant
ajouter un nom en fin de liste
supprimer le nom courant
supprimer la liste
initialiser la liste

e] les deux listes contenant les noms des attributs des entités de travail:

Pour chaque liste:

se positionner en début de liste
lire le nom courant
passer à l'élément suivant
ajouter un nom en fin de liste
supprimer le nom courant
supprimer la liste
initialiser la liste

f] la liste contenant les correspondances déja détectées entre les schémas de travail:

se positionner en début de liste
lire le nom courant
passer à l'élément suivant
ajouter un nom en fin de liste
supprimer le nom courant
supprimer la liste
initialiser la liste

g] la table de correspondance entre les noms des schémas et les noms des fichiers les contenant:

se positionner en début de liste
donner le couple courant
passer au couple suivant
ajouter un couple en fin de liste
supprimer le couple courant
supprimer la liste
initialiser la liste

Définition des fenêtres de notre application

```
BEGIN MAIN

Data type definition:
BEGIN
 LISTE "ListEnt"
   begin
     "nom";
   end
 LISTE "ListAtt"
   begin
     "nom";
    "cardinalité";
     "fils";
   end
 LISTE "ListCorr"
   begin
     "automatique";
     "schema1";
     "schema2";
     "objet1";
     "objet2";
     "type";
     "objet";
  end
 LISTE "ListSch"
   begin
     "nom schema";
     "nom fichier";
   end
END

Data definition:
BEGIN
  ListSch"SchFile";{table de correspondance fichier/schema}
  STRING "SchSel1";{premier schema selectionne}
  STRING "SchSel2";{second schema selectionne}
  STRING "FileSel1";{fichier correspondant au premier schema selectionne}
  STRING "FileSel2";{fichier correspondant au second schema selectionne}
  STRING "EntSel1"; {première entité selectionnée}
  STRING "EntSel2"; {seconde entité selectionnée}
  ListEnt "EntList1";{liste des entités du premier schema}
  ListEnt "EntList2"; {liste des entités du second schema}
  ListAtt "AttList1";{liste des attributs de la première entité}
  ListAtt "AttList2"; {liste des attributs de la seconde entité}
  ListCorr "Correspondances";{liste des correspondances entre les 2 sch}
  STRING "Object";
  TEXT "Descr";
  TEXT "HelpText";{texte d'aide à l'iutilisateur}
END
```

DEF_WINDOW "Tiled"

Consists of:

BEGIN

```
 Menu  M1

  begin
     item "Help"
     PopUp "Select"
        begin
           Item "Schema";
           Item "Entity";
           Item "Attribute";
        end
    PopUp "Descr"
      begin
         Item "Sch1";
         Item "Sch2";
         Item "Ent1";
         Item "Ent2";
         Item "Att1";
         Item "Att2";
       end
   PopUp "Tree"
     begin
        Item "Tree1";
        Item "Tree2";
      end
end;
Client area;
```

END

Receives Messages:

BEGIN

```
  PAINT from: system
    performs: display
CLICK MENUITEM "Help"
    perform: SendMessage(HelpWindow, PAINT);
CLICK POPUPITEM "Schema" from "Select"
    perform: SendMessage(SelectSchema, PAINT);
CLICK POPUPITEM "Entity" from "Select"
    perform: SendMessage(SelectEntity, PAINT);
CILCK POPUPITEM "Attribute" from "Select"
    perform: Sendmessage(SelectAttribute, PAINT);
CLICK POPUPITEM "Sch1" from "Descr"
    perform: SendMessage(DescrWindow, PAINT1);
CLICK POPUPITEM "Sch2" from "Descr"
    perform: SendMessage(DescrWindow, PAINT2);
```

```
CLICK POPUPITEM "Ent1" from "Descr"
     perform: SendMessage(DescrWindow, PAINT3);
CLICK POPUPITEM "Ent2" from "Descr"
     perform: SendMessage(DescrWindow, PAINT4);
CLICK POPUPITEM "Att1" from "Descr"

     perform: SendMessage(DescrWindow, PAINT5);
CLICK POPUPITEM "Att2" from "Descr"
     perform: SendMessage(DescrWindow, PAINT6);
CLICK POPUPITEM "Tree1" from "Tree"
     perform: SendMessage(TreeWindow, PAINT1);
CLICK POPUPITEM "Tree2" from "Tree"
     perform: SendMessage( TreeWindow, PAINT2);
CLICK CLOSE
      perform: Call CloseWindow;

END
DATA
BEGIN
END

DEF-WINDOW "SelectSchema"

consists of:

BEGIN
  ListBox "Schema";
  BUTTON "Ok";
END

Receives Messages:

BEGIN

PAINT from Tiled Window
  perform:  display SchFile;
CLICK LISTBOX "Schema";
 perform: SchSel1=CHOICE;
          || SchSel2=CHOICE;
CLICK BUTTON "Ok"
      perform:   Call CloseWindow;
END

Data:

BEGIN
   SchFile;
   SchSel1;
   SchSel2;
   FileSel1;
   FileSel2;
   EntList1;
   EntList2;
END
```

```
DEF-WINDOW "SelectEntity"

consists of:

BEGIN

   LISTBOX "Entity 1";
   LISTBOX "Entity 2";
   LISTBOX "Propositions";
   BUTTON "Ok";

END

Receives Messages:

BEGIN

 PAINT from Tiled Window
    perform: display EntList1&&EntList2;
 CLICK LISTBOX "Entity1";
    perform: EntSel1= CHOICE
 CLICK LISTBOX "Entity2";
    perform : EntSel 2= CHOICE
 CLICK BUTTON "Ok"
    perform: Call CloseWindow;

data:

BEGIN

   FileSel1;
   FileSel2;
   EntList1;
   EntList2;
   EntSel1;
   EntSel2;
   AttList1;
   AttList2;
  Correspondances;

END
```

DEF-WINDOW "SelectAttribute"

consists of:

BEGIN

LISTBOX "Attribute1";
LISTBOX "Attribute2";
LISTBOX "Propositions";
BUTTON "Ok";

END

Receives Messages:

BEGIN

PAINT from Tiled Window
perform: display
CLICK LISTBOX "Attribute 1"
perform: AttSel1= CHOICE
CLICK LISTBOX "Attribute 2"
perform: AttSel2= CHOICE
CLICK LISTBOX "Propositions"
perform:
CLICK BUTTON "Ok"
perform: Call CloseWindow;

END

Data:

BEGIN

FileSel1;
FileSel2;
AttSel1;
AttSel2;
Correspondances;

END

DEF-WINDOW "TreeWindow"

consists of:

BEGIN

Client Area;

END

Receives Messages

```
BEGIN

 PAINT from Tiled Window
    perform: display AttList1;
 PAINT from Tiled Wiondow
    perform: display AttList2;
 CLICK CLOSE
    perform: Call CloseWindow;

END
```

Data:

```
BEGIN
   Attlist1||AttList2;
END
```

DEF-WINDOW "DescrWindow"

consists of:

```
BEGIN
  Client Area;
END
```

Receives Messages:

```
BEGIN
 PAINT from Tiled Window
     perform: Object= SchSel1;
                 display Descr(Object);
 PAINT from Tiled Window
    perform: Object= SchSel2;
                display Descr(Object);
 PAINT3 from Tiled Window
    perform:=Object= EntSel1;
                display Descr(Object);
 PAINT4 from Tiled Window
    perform: Object= Entsel2;
                display Descr(Object);
 PAINT 5 from Tiled Window
    perform: Object= AttSel1;
                display Descr(Object);
 PAINT 6 from Tiled Window
    perform: Object= AttSel2;
                display Descr(Object);
 CLICK CLOSE
    perform: Call CloseWindow;
 END
```

```
Data:

BEGIN

  File1||File2;
  SchSel1||SchSel2||EntSel1||EntSel2||AttSel1||AttSel2;

END

DEF-WINDOW "HelpWindow"

consists of:

BEGIN

  Client Area;

END

Receives Messages:

BEGIN

  PAINT from Tiled Window
    perform: display HelpText;
  CLICK CLOSE
    perform: Call CloseWindow;

END

Data:

BEGIN

  HelpText;

END

END MAIN
```

```
/*****************************************************************************/
/*                                                                           */
/*                          LE FICHIER DES RESSOURCES                        */
/*                                                                           */
/*****************************************************************************/

/* menu de la tiled window */
principal MENU
begin
     MENUITEM "Help"       , ID_HELP
     POPUP "Select"
       BEGIN
          MENUITEM "Schema" , ID_SELECTSCH
          MENUITEM "Entity" , ID_SELECTENT
          MENUITEM "Attribute" , ID_SELECTATTR
       END
      POPUP "Descr"
        BEGIN
          MENUITEM "Schema_1" , ID_DESCRSCH1
          MENUITEM "Schema_2" , ID_DESCRSCH2
          MENUITEM "Entity_1" , ID_DESCRENT1
          MENUITEM "Entity_2" , ID_DESCRENT2
          MENUITEM "Attr_1" , ID_DESCRATTR1
          MENUITEM "Attr_2" , ID_DESCRATTR2
        END
     POPUP "Tree"
       BEGIN
         MENUITEM "Tree1" , ID_TREE1
         MENUITEM "Tree2" , ID_TREE2
       END
     POPUP "Write"
       BEGIN
         MENUITEM "Corr EE" , ID_CORRENT
         MENUITEM "Corr AA" , ID_CORRATT
       END
end

/*****************************************************************************/
/* selecction d'un type de correspondance EE */

CORRENT DIALOG LOADONCALL MOVEABLE DISCARDABLE 15, 19, 139, 120
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
CONTROL "Choose corr. between ent." -1, "static", SS_CENTER
| WS_CHILD, 10, 9, 118, 8
CONTROL "same population" RB1, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 51, 29, 82, 12
CONTROL "subtype of"RB2, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 50, 51, 82, 12
CONTROL "surtype of"RB9, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 49, 73, 82, 12
CONTROL "common surtype"RB3, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD,50,95,82,12
CONTROL "Ok" ID_OK, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 10, 51, 24, 14
CONTROL "Cancel" ID_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 10, 73, 24, 14
END
```

```
/*****************************************************************************/
/* selection d'un type de correspondance AA */

CORRATT DIALOG LOADONCALL MOVEABLE DISCARDABLE 16, 23, 139, 127
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
CONTROL "choose corr. between att." -1, "static", SS_LEFT
| WS_CHILD, 11, 5, 118, 8
CONTROL "identity" RB4, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 50, 63, 82, 12
CONTROL "more precise" RB5, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 50, 85, 82, 12
CONTROL "less precise" RB6, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 49, 107, 82, 12
CONTROL "OK" ID_OK, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 10, 85, 24, 14
CONTROL "Cancel" ID_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 10, 107, 30, 14
CONTROL "point of view" RB7, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 50, 23, 82, 12
CONTROL "function" RB8, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 50, 43, 82, 12
END

/*****************************************************************************/
/* selection de deux entites par list-box */

LISTBOX DIALOG LOADONCALL MOVEABLE DISCARDABLE 13, 13, 242, 149
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
   CONTROL "Text" ID_LB_1, "listbox", LBS_NOTIFY | LBS_STANDARD
| WS_BORDER | WS_VSCROLL | WS_CHILD, 10, 20, 95, 41

   CONTROL "Text" ID_LB_2, "listbox", LBS_NOTIFY | LBS_STANDARD
| WS_BORDER | WS_VSCROLL | WS_CHILD, 136, 20, 95, 41

   CONTROL "Text" ID_LB_3, "listbox", LBS_NOTIFY | LBS_STANDARD
| WS_BORDER | WS_VSCROLL | WS_CHILD, 10, 85, 222, 41

   CONTROL "OK" ID_OK, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 138, 132, 33, 14

   CONTROL "Cancel" ID_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 70, 132, 34, 14

   CONTROL "ENTITY LIST 1" -1, "static", SS_CENTER | WS_CHILD, 11, 3, 85, 13

   CONTROL "ENTITY LIST 2" -1, "static", SS_CENTER
| WS_CHILD, 136, 3, 88, 12

   CONTROL "PROPOSITIONS" -1, "static", SS_CENTER | WS_CHILD, 11, 67, 214, 14

END

/*****************************************************************************/
/* selection de deux attributs par list-box */

ATTSEL DIALOG LOADONCALL MOVEABLE DISCARDABLE 13, 13, 242, 149
STYLE WS_DLGFRAME | WS_POPUP
```

```
BEGIN
   CONTROL "Text" ID_LBATT_1, "listbox", LBS_NOTIFY | LBS_STANDARD
| WS_BORDER | WS_VSCROLL | WS_CHILD, 10, 20, 95, 41

   CONTROL "Text" ID_LBATT_2, "listbox", LBS_NOTIFY | LBS_STANDARD
| WS_BORDER | WS_VSCROLL | WS_CHILD, 136, 20, 95, 41

   CONTROL "Text" ID_LBATT_3, "listbox", LBS_NOTIFY | LBS_STANDARD
| WS_BORDER | WS_VSCROLL | WS_CHILD, 10, 85, 222, 41

   CONTROL "Ok" ID_OK, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 138, 132, 33, 14

   CONTROL "Cancel" ID_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP
| WS_CHILD, 70, 132, 34, 14

   CONTROL "ATTRIBUTE LIST 1" -1, "static", SS_CENTER | WS_CHILD, 11, 3, 85, 13

   CONTROL "ATTRIBUTE LIST 2" -1, "static", SS_CENTER
| WS_CHILD, 136, 3, 88, 12

   CONTROL "PROPOSITIONS" -1, "static", SS_CENTER | WS_CHILD, 11, 67, 214, 14

END

/*****************************************************************************/
/* introduction des noms de fichiers contenant les schemas */

EDT DIALOG LOADONCALL MOVEABLE DISCARDABLE 11, 19, 211, 91
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
   CONTROL "Enter file name" -1, "static", SS_LEFT | WS_CHILD, 32, 7, 140, 14
   CONTROL "" ID_FILENAME, "edit", ES_LEFT | WS_BORDER
          | WS_TABSTOP | WS_CHILD, 16, 37, 178, 19
   CONTROL "Next" ID_OK, "button", BS_PUSHBUTTON | WS_TABSTOP
          | WS_CHILD, 51, 72, 34, 14
   CONTROL "End" ID_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP
          | WS_CHILD, 119, 72, 38, 14
END

/*****************************************************************************/
/* selection d'un schema par list-box */

SCHEMAS DIALOG LOADONCALL MOVEABLE DISCARDABLE 12, 21, 250, 80
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
   CONTROL "Select a schema" -1, "static", SS_LEFT | WS_CHILD, 7, 7, 70, 10

   CONTROL "Propositions" -1, "static", SS_LEFT | WS_CHILD, 84, 7, 111, 11

   CONTROL "Text" LB_SCH1, "listbox", LBS_NOTIFY
 | LBS_STANDARD | WS_BORDER | WS_VSCROLL | WS_CHILD, 6, 23, 70, 50

   CONTROL "Text" LB_SCH2, "listbox", LBS_NOTIFY
 | LBS_STANDARD | WS_BORDER | WS_VSCROLL | WS_CHILD, 84, 23, 113, 50

/* CONTROL "Descr" ID_DESCR, "button", BS_PUSHBUTTON
| WS_TABSTOP | WS_CHILD, 200, 25, 36, 14 */
```

```
CONTROL "OK" ID_OK, "button", BS_PUSHBUTTON
| WS_TABSTOP | WS_CHILD, 207, 55, 24, 14

END

/*****************************************************************************/
```

```
/*************************************************************************/
/* T H E P R O G R A M'S E X T E R N A L V A R I A B L E S */
/*************************************************************************/

typedef struct elem          /* structure de l'arbre des attributs */
{  char  name[31];
    char ConMin[2];
    char ConMax[2];
    struct elem FAR *son;     /* fils le plus a gauche dans l'arbre */
    struct elem FAR *brother; /* frere de droite dans l'arbre */
} *PELEM, FAR *LPELEM;

typedef struct lcorrelem
{ char dauto[2];                /* detectee automatiquement ou non */
    char s1 [50];               /* schema 1 */
    char s2[50];                /* schema 2 */
    char obj[3];                /* EE,AA,CS,MS */
    char type[3];
    char n1[50];                /* nom de l'objet 1 */
    char n2[50];                /* nom de l'objet 2 */
    struct lcorrelem FAR *next;
}  FAR *LCORR;

typedef struct lelem
{ char name[50];
  struct lelem FAR *next;
} FAR  *LISTE;

typedef struct lschelem     /* table de correspondance fichier/schema */
{ char filen[50];           /* nom de fichier DSL */
  char schn[50];            /* nom de schema DSL */
  struct lschelem FAR *next;
} FAR *LISTESCH;

/* static */char factif[] =
          "initialisation du nom de fichier factif .....FIN";
/* static */long stoffs=0;
/* static */char stbuf[LBLOCK];

/* static */ HANDLE    hInst;

HWND hhh;
HWND hhhh;

HWND hC1Wnd = NULL;       /* handle of the HELP window */
HWND hC2Wnd = NULL;       /* handle of the DESCR window */
HWND hC3Wnd = NULL;       /* handle of the TREE window */

HWND hC5Wnd = NULL;       /* handle of the PARAM window */

/* static */char szLBOutput[80]; /* store the listbox selection */
/* static */char szLBOutput1[80];
/* static */char szLBOutput2[80];

/* static */char szLBOutputAtt1[80]; /* ListBox de selectatt */
/* static */char szLBOutputAtt2[80];

LPELEM liste2; /* liste des mots a afficher dans la 2e listbox */
LISTESCH listsch = NULL; /* liste des fichiers from listschemas */
```

```
LCORR ListCorr = NULL;
char filename[50];

WORD corrent = RB1;
WORD corratt = RB4;

BOOL firstselect = TRUE;
BOOL firstattr = TRUE;

char File1[50];            /* fichiers de travail */
char  File2[50];
char Schema1[50];          /* schemas de travail */
char Schema2[50];
char EntSel1[50];          /* entites de travail */
char EntSel2[50];
char AttSel1[50];          /* attributs de travail */
char AttSel2[50];

/* AttSel1 appartient a EntSel1 qui appartient a Schema1 qui se trouve
dans File1 */

LISTE LAtt1=NULL;          /* liste (non arborescente des attr de EntSel1 */
LISTE LAtt2=NULL;
LCORR lcorrAA1 = NULL;
LCORR lcorrAA2 = NULL;
LISTE ListEnt1 = NULL; /* liste des entites de Schema1 */
LISTE ListEnt2 = NULL;


FARPROC lpEditWndProc;   /* var de ED PROCEDURE       */
HANDLE hTextEntry;
WORD  TextHandle;
HANDLE hEdtextWnd;
int  xTextExt  =  200;
int  yTextExt  =  100;



int HelpOrgX = 0; /* position of the help window */
int HelpOrgY = 0 ;
int Gap;              /* length of a vertical scrolling */
int PageH;            /* length of a page up/down     */
int PageW;            /* width                        */



int xTextExt;          /*** var essai de l'ed ***/
int yTextExt;
HWND hTextEntry;
HANDLE TextHandle;
HWND hEdtextWnd;
HANDLE hDlg;
FARPROC lpEditWndProc;
HANDLE hWaitCurs;
HANDLE hArrowCurs;
int count;
RECT TilWnd;          /*  Tiled  window's  size  */
```

```
LPELEM Tree1; /* pointeur vers la racine de l'arbre des attributs de EntSel1 */
LPELEM Tree2; /* pointeur vers la racine de l'arbre des attributs de EntSel2 */

char *entier;
int TreeOrgX = 0; /* position de la TREE window */
int TreeOrgY = 0;
int FontW = 0;       /* largeur d'un caractere */
int MaxBrother = 0; /* coordonee en X maximale d'un attribut de l'arbre */
BOOL EntiteGlob;     /* utilise pour determine si l'on affiche la racine de*/
                     /* l'arbre ou un des attributs                        */


/*****************************************************************************/
/*              F O R W A R D R E F E R E N C E S                            */
/*****************************************************************************/


void PRT_STR (PSTR, ...);
BOOL FAR PASCAL InitInteg  ( HANDLE, HANDLE, int)        ;
BOOL FAR PASCAL InitIntegFirst ( HANDLE)                 ;
BOOL FAR PASCAL InitIntegEvery ( HANDLE, int)            ;
BOOL FAR PASCAL CloseInteg ( HANDLE)                     ;
BOOL FAR PASCAL PaintIntegWindow (HWND)                  ;
LONG FAR PASCAL IntegWindowProc( HWND, unsigned, WORD, LONG);
FAR PASCAL processmenu(HWND,WORD);

BOOL FAR PASCAL ListBoxWindowProc (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL AttSelWindowProc (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Edtproc (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL HelpSetScroll(HWND);
BOOL FAR PASCAL PaintHelpWindow (HWND);
FAR PASCAL ProcessHelp(HWND);
BOOL FAR PASCAL ListSchemas(HWND);
BOOL FAR PASCAL SelectSchemas(HWND);
BOOL FAR PASCAL AutoAid(HWND,LISTE,LISTE,PSTR);
BOOL FAR PASCAL InterAid(HWND);
BOOL FAR PASCAL Transit();
BOOL FAR PASCAL PreciseEnt();
FAR PASCAL ProcessDescr(HWND);
LONG FAR PASCAL HelpProc(HWND, unsigned, WORD, LONG)  ;
BOOL FAR TextOpen (LPSTR);
BOOL FAR InitInstance(HWND,WORD,WORD,WORD,WORD);
void BlankTextBuffer();
void SetTextEntry(LPSTR);
LONG FAR PASCAL TextEntryProc(HWND,unsigned,WORD,LONG);
BOOL FAR PASCAL PaintTreeWindow (HWND,LPELEM);
BOOL FAR PASCAL HScrolling(HWND,WORD,LONG);
BOOL FAR PASCAL VScrolling(HWND,WORD,LONG);
LONG FAR PASCAL TreeWindowProc1 (HWND,unsigned,WORD,LONG);
LONG FAR PASCAL TreeWindowProc2 (HWND,unsigned,WORD,LONG);
LONG FAR PASCAL ParamWindowProc (HWND,unsigned,WORD,LONG);
FAR PASCAL ProcessTree1(HWND);
FAR PASCAL ProcessTree2(HWND);
BOOL FAR PASCAL TreeSetScroll(HWND);
LONG FAR PASCAL DescrWndProc(HWND, unsigned, WORD, LONG);
LISTE FAR PASCAL add(LPSTR, LISTE, HWND);
HANDLE listalloc(HWND);
FAR PASCAL ltest(LISTE,HWND);
LPSTR FAR PASCAL CorrSch(LONG);
/*FAR PASCAL*/ CopyDescr(HWND,PSTR,LONG);
/*FAR PASCAL*/ InitDescrSchema(HWND,PSTR);
```

```
BOOL FAR PASCAL LBschWindowProc(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL SelectSchemas(HWND);
long /*FAR PASCAL*/ findobject(PSTR,long,PSTR);
long/* FAR PASCAL*/ readblock(PSTR,long, LPSTR);
int /*FAR PASCAL*/ readcar(PSTR,long);
LISTE FAR PASCAL ConvertTree(HWND,LPELEM,LISTE,BOOL);
/* LCORR */ FAR PASCAL IdEnt(HWND,LISTE,LISTE,PSTR);
/* LCORR */ FAR PASCAL SynEnt(HWND,LISTE,LISTE,PSTR);
BOOL FAR PASCAL ReadSyn(LPSTR,LPSTR);
FAR PASCAL InitSdc(HWND);
HANDLE FAR PASCAL lcalloc(HWND);
BOOL FAR PASCAL correntproc(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL corrattproc(HWND, unsigned, WORD, LONG);
LONG FAR PASCAL findname(PSTR, LONG, PSTR);
LPSTR FAR PASCAL CorrFile(LONG);
BOOL FAR PASCAL AttSel(HWND);
LISTE FAR PASCAL conslist(HWND, PSTR, PSTR);
LONG FAR PASCAL FindPostDef(PSTR, LONG, PSTR);
PSTR PosSch(PSTR);
PrintTree(HWND, LPELEM, HDC, int, int, int, int);
HANDLE ListSchAlloc(HWND);
FAR PASCAL AddSch(HWND, PSTR);
void  PRT_STR(PSTR, ...);
BOOL FAR fcopy(HWND, PSTR, PSTR);
BOOL FAR wblock(HWND, PSTR, LPSTR, int, LONG);
PSTR FAR PASCAL CutList(PSTR);
HANDLE FAR PASCAL TreeAlloc(HWND);
InitArbre(HWND, PSTR, LONG, LPELEM);
LPELEM InitTree(HWND, PSTR, PSTR);
LONG GroupElem(PSTR, PSTR, LONG);
LONG PosEntity(PSTR, PSTR, LONG);
InitDescrEnt(HWND, PSTR, PSTR);
InitDescrAtt(HWND, PSTR, PSTR);
InitDescr(HWND);
LCORR alc(PSTR, LCORR, HWND);
LCORR clc(PSTR, PSTR);
LCORR aelc(LCORR, PSTR, PSTR, PSTR, PSTR, PSTR, LPSTR, LPSTR);
BOOL wlc(LCORR);
BOOL wfirstlc(LCORR);
ProcessParam(HWND);
BOOL FAR PASCAL PaintParamWindow(HWND);
int  strcmp();
int  strcpy();
int   close();
int  open();
int  strcat();
int  read();
int  write();
BOOL ffc(PSTR, PSTR);
```

```
/*****************************************************************************/
/*                                                                           */
/*                              W I N M A I N                                */
/*                                                                           */
/*****************************************************************************/
/* Contient le programme principal et les procedures d'initialisation et
de cloture imposees par Windows */


/*****************************************************************************/
/*                          I N C L U D E F I L E S                          */
/*****************************************************************************/

/*****************************************************************************/
/*                        M A I N P R O C E D U R E                          */
/*****************************************************************************/


int PASCAL WinMain ( hInstance ,
                     hPrevInstance,
                     lpszCmdLine ,
                     CmdShow     )

HANDLE hInstance, hPrevInstance;
LPSTR lpszCmdLine            ; /* LPSTR to the command line        */
int CmdShow                  ; /* Iconic or tiled at start          */

{
MSG msg               ; /* Temp buffer to hold message       */

InitInteg ( hInstance, hPrevInstance, CmdShow);
while (GetMessage((LPMSG)&msg, NULL, 0, 0))
{
  TranslateMessage ((LPMSG)&msg);
  DispatchMessage ((LPMSG)&msg);
        }
        CloseInteg ( hInstance);
        return (msg.wParam);
}


/*****************************************************************************/
/*                        I N I T I A L I S A T I O N                        */
/*****************************************************************************/


BOOL FAR PASCAL InitInteg ( hInstance, hPrevInstance, CmdShow)

HANDLE hInstance, hPrevInstance;
int  CmdShow;

{
if (!hPrevInstance)
    InitIntegFirst (hInstance);
InitIntegEvery ( hInstance, CmdShow);
return TRUE;
}
```

```
/*************************************************************************/
/* F I R S T - I N S T A N C E I N I T I A L I S A T I O N             */
/*************************************************************************/
/* Definition des classes de fenêtres */

BOOL FAR PASCAL InitIntegFirst (hInstance)

HANDLE hInstance;

{
WNDCLASS wcIntegClass;
WNDCLASS wcChildClass;

/* Tilled Window */
wcIntegClass.lpszClassName = (LPSTR) "Integ";
wcIntegClass.hInstance = hInstance                              ;
wcIntegClass.lpfnWndProc = IntegWindowProc                    ;
wcIntegClass.hCursor     = LoadCursor (NULL, IDC_ARROW)       ;
wcIntegClass.hIcon       = LoadIcon (NULL,IDI_APPLICATION)    ;
wcIntegClass.lpszMenuName = (LPSTR) "principal"                          ;
wcIntegClass.hbrBackground = GetStockObject (WHITE_BRUSH)     ;
wcIntegClass.style       = CS_HREDRAW | CS_VREDRAW           ;
wcIntegClass.cbClsExtra = 0                                     ;
wcIntegClass.cbWndExtra = 0                                     ;

RegisterClass ((LPWNDCLASS) &wcIntegClass);

/* Help Window */
wcChildClass.lpszClassName = (LPSTR) "HELP"                     ;
wcChildClass.hInstance = hInstance                              ;
wcChildClass.lpfnWndProc = HelpProc                             ;
wcChildClass.hCursor     = LoadCursor (NULL, IDC_ARROW)         ;
wcChildClass.hIcon       = LoadIcon (NULL,IDI_APPLICATION)      ;
wcChildClass.lpszMenuName = (LPSTR) NULL                        ;
wcChildClass.hbrBackground = GetStockObject (WHITE_BRUSH)      ;
wcChildClass.style       = CS_HREDRAW | CS_VREDRAW              ;
wcChildClass.cbClsExtra = 0                                     ;
wcChildClass.cbWndExtra = 0                                     ;

RegisterClass ((LPWNDCLASS) &wcChildClass);

/* Descr Window */
wcChildClass.lpszClassName = (LPSTR) "DESCR"                    ;
wcChildClass.hInstance = hInstance                              ;
wcChildClass.lpfnWndProc = DescrWndProc                         ;
wcChildClass.hCursor     = LoadCursor (NULL, IDC_ARROW)         ;
wcChildClass.hIcon       = LoadIcon (NULL,IDI_APPLICATION)      ;
wcChildClass.lpszMenuName = (LPSTR) NULL                        ;
wcChildClass.hbrBackground = GetStockObject (WHITE_BRUSH)      ;
wcChildClass.style       = CS_HREDRAW | CS_VREDRAW              ;
wcChildClass.cbClsExtra = 0                                     ;
wcChildClass.cbWndExtra = 0                                     ;

RegisterClass ((LPWNDCLASS) &wcChildClass);

/* Tree1 Window */
wcChildClass.lpszClassName = (LPSTR) "TREE1"                    ;
wcChildClass.hInstance = hInstance                              ;
wcChildClass.lpfnWndProc = TreeWindowProc1                      ;
```

```
wcChildClass.hCursor     = LoadCursor (NULL, IDC_ARROW)       ;
wcChildClass.hIcon       = LoadIcon (NULL,IDI_APPLICATION)    ;
wcChildClass.lpszMenuName = (LPSTR) NULL                      ;
wcChildClass.hbrBackground = GetStockObject (WHITE_BRUSH)     ;
wcChildClass.style       = CS_HREDRAW | CS_VREDRAW            ;
wcChildClass.cbClsExtra = 0                                   ;
wcChildClass.cbWndExtra = 0                                   ;

RegisterClass ((LPWNDCLASS) &wcChildClass);

/* Tree2 Window */
wcChildClass.lpszClassName = (LPSTR) "TREE2"                  ;
wcChildClass.hInstance = hInstance                            ;
wcChildClass.lpfnWndProc = TreeWindowProc2                    ;
wcChildClass.hCursor     = LoadCursor (NULL, IDC_ARROW)       ;
wcChildClass.hIcon       = LoadIcon (NULL,IDI_APPLICATION)    ;
wcChildClass.lpszMenuName = (LPSTR) NULL                      ;
wcChildClass.hbrBackground = GetStockObject (WHITE_BRUSH)     ;
wcChildClass.style       = CS_HREDRAW | CS_VREDRAW            ;
wcChildClass.cbClsExtra = 0                                   ;
wcChildClass.cbWndExtra = 0                                   ;

RegisterClass ((LPWNDCLASS) &wcChildClass);

/* Param Window */
wcChildClass.lpszClassName = (LPSTR) "PARAM"                  ;
wcChildClass.hInstance = hInstance                            ;
wcChildClass.lpfnWndProc = ParamWindowProc                    ;
wcChildClass.hCursor     = LoadCursor (NULL, IDC_ARROW)       ;
wcChildClass.hIcon       = LoadIcon (NULL,IDI_APPLICATION)    ;
wcChildClass.lpszMenuName = (LPSTR) NULL                      ;
wcChildClass.hbrBackground = GetStockObject (WHITE_BRUSH)     ;
wcChildClass.style       = CS_HREDRAW | CS_VREDRAW            ;
wcChildClass.cbClsExtra = 0                                   ;
wcChildClass.cbWndExtra = 0                                   ;

RegisterClass ((LPWNDCLASS) &wcChildClass);

        return TRUE;
}



/*************************************************************************/
/* E V E R Y - I N S T A N C E I N I T I A L I S A T I O N       */
/*************************************************************************/


BOOL FAR PASCAL InitIntegEvery ( hInstance, CmdShow)

HANDLE hInstance;
int  CmdShow  ;

{
HWND hWnd;

hInst = hInstance;
hWnd = CreateWindow((LPSTR)  "Integ", /* Window class name    */
```

```
                (LPSTR)  "AIDE A LA DETECTION DE CORRESPONDANCES INTER SCHEMAS",
                    WS_TILEDWINDOW|WS_CLIPCHILDREN,
                    0                   , /* x - ignored for tiled */
                    0                   , /* y - ignored for tiled */
                    0                   , /* cx - ignored for tiled */
                    0                   , /* cy - ignored for tiled */
                    (HWND)NULL          , /* No parent for this wind */
                    (HMENU)NULL         , /* Use the class menu      */
                    (HANDLE)hInstance , /* Who created this window */
                    (LPSTR)NULL           /* No params. to pass on.  */
                   );
ShowWindow (hWnd, CmdShow);
hhh=hWnd;
hhhh=hhh;
SendMessage(hWnd, CREATION, (WORD) 1, (LONG) 2 );
UpdateWindow (hWnd);
return TRUE;
}


/*************************************************************************/
/*                  C L O S I N G R O U T I N E                          */
/*************************************************************************/

BOOL FAR PASCAL CloseInteg (hInstance)

HANDLE hInstance;

{
 wlc(ListCorr);        /* Sauve les correspondances */
 return TRUE;
}

/*************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*          T H E T I L E D W I N D O W P R O C E D U R E                    */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                              INTEGWINDOWPROC                              */
/*****************************************************************************/
/* procedure de la fenetre principale qui gere les messages envoyes par Windows
*/
long FAR PASCAL IntegWindowProc (hWnd,
                                 message,
                                 wParam,
                                 lParam)

HWND     hWnd  ;
unsigned message;
WORD     wParam ;
LONG     lParam ;

{
static char buffer[50] = "                                                  " ;

switch (message)
{

case WM_CREATE :
                 return 0L;

case  CREATION  :


                 firstselect=TRUE;
                 lstrcpy(File1,(LPSTR)"");
                 lstrcpy(File2,(LPSTR)"");
                 lstrcpy(EntSel1,(LPSTR)"");
                 lstrcpy(EntSel2,(LPSTR)"");

                 ListSchemas(hWnd);
                                 /* choose n schemas to work on */
                 InitSdc(hhh);
                 ListCorr=NULL;
                                 /* mise en place des "grisés" */
          EnableMenuItem(GetMenu(hWnd),ID_SELECTENT,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_SELECTATTR,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRSCH1,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRSCH2,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRENT1,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRENT2,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRATTR1,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRATTR2,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_TREE1,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_TREE2,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_CORRENT,MF_DISABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_CORRATT,MF_DISABLED|MF_BYCOMMAND);

              break;
```

```
case WM_DESTROY :
                PostQuitMessage (0);
                break;

case WM_PAINT :
                PaintIntegWindow (hWnd);
                break;

case WM_COMMAND:
                processmenu(hWnd,message,wParam,lParam);
                break;


default :
                return ( DefWindowProc(hWnd, message, wParam,
                        lParam));
                break;
}
return (0L);
}


/*****************************************************************************/
/*                T H E P A I N T P R O C E D U R E                          */
/*****************************************************************************/


BOOL FAR PASCAL PaintIntegWindow (hWnd)

HWND    hWnd  ;

{
PAINTSTRUCT ps ;
HDC         hDC;

BeginPaint (hWnd, (LPPAINTSTRUCT)  &ps);
hDC = ps.hdc;
ValidateRect (hWnd, (LPRECT)NULL);
EndPaint (hWnd, (LPPAINTSTRUCT) &ps);

return TRUE;

}

/*****************************************************************************/
/*                T H E PROCESSMENU P R O C E D U R E                        */
/*****************************************************************************/
/* traite les commandes de l'utilisateur issues d'une sélection menu dans la
tiled Window */

FAR PASCAL processmenu(hWnd,message,wId,lParam)


HWND hWnd;
unsigned message;
WORD wId;
LONG lParam;

{
```

```
OFSTRUCT ofs1, ofs2;
int fromf;
FARPROC lpa;
char  Sc1[50];
char  Sc2[50];
static LISTE LA1Save,LA2Save;

switch (wId)
     {
      case ID_HELP:
            ProcessHelp(hWnd);
            break;

      case ID_SELECTSCH:

            if (ListCorr != NULL)
             {
              wlc(ListCorr);
             };
            if(firstselect)
             {
              lstrcpy(EntSel1,(LPSTR)"");
              lstrcpy(AttSel1,(LPSTR)"");
              Tree1=NULL;
              SelectSchemas(hWnd);
              firstselect=FALSE;
              ListEnt1= conslist(hWnd, (PSTR) "ENTITY",File1);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRSCH1,MF_ENABLED|MF_BYCOMMAND);
              if (ListCorr != NULL)
               {
                lstrcpy((LPSTR)Sc1,(LPSTR)Schema1);
                lstrcpy((LPSTR)Sc2,(LPSTR)Schema2);
                ListCorr=NULL;
                ListCorr=clc((PSTR)Sc1,(PSTR)Sc2);
                AutoAid(hhh,ListEnt1,ListEnt2,"EE");
               };
             }
            else
             {
              lstrcpy(EntSel2,(LPSTR)"");
              lstrcpy(AttSel2,(LPSTR)"");
              Tree2=NULL;
              SelectSchemas(hWnd);
              firstselect=TRUE;
              ListEnt2= conslist(hWnd, (PSTR) "ENTITY",File2);
          EnableMenuItem(GetMenu(hWnd),ID_DESCRSCH2,MF_ENABLED|MF_BYCOMMAND);
          EnableMenuItem(GetMenu(hWnd),ID_SELECTENT,MF_ENABLED|MF_BYCOMMAND);
              lstrcpy((LPSTR)Sc1,(LPSTR)Schema1);
              lstrcpy((LPSTR)Sc2,(LPSTR)Schema2);
              ListCorr=NULL;
              ListCorr=clc((PSTR)Sc1,(PSTR)Sc2);
              AutoAid(hhh,ListEnt1,ListEnt2,"EE");
             };
            ProcessParam(hhh);
            break;

      case ID_SELECTENT:
           EnableMenuItem(GetMenu(hWnd),ID_DESCRENT1,MF_DISABLED|MF_BYCOMMAND);
           EnableMenuItem(GetMenu(hWnd),ID_TREE1,MF_DISABLED|MF_BYCOMMAND);
```

```
        EnableMenuItem(GetMenu(hWnd),ID_DESCRENT2,MF_DISABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_TREE2,MF_DISABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_SELECTATTR,MF_DISABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_CORRENT,MF_DISABLED|MF_BYCOMMAND);
        AlgoCorr(LA1Save,LA2Save);
        Tree1=NULL;
        Tree2=NULL;
        LAtt1=NULL;
        LAtt2=NULL;
        lstrcpy((LPSTR)AttSel1,(LPSTR)"");
        lstrcpy((LPSTR)AttSel2,(LPSTR)"");
        InterAid(hWnd);
        if(lstrcmp(EntSel1,(LPSTR)"")!=0)
            {
        EnableMenuItem(GetMenu(hWnd),ID_DESCRENT1,MF_ENABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_TREE1,MF_ENABLED|MF_BYCOMMAND);
             Tree1=InitTree(hhh,File1,EntSel1);
             LAtt1=ConvertTree(hhh,Tree1,LAtt1,TRUE);
             LA1Save=LAtt1;
            };
        if(lstrcmp(EntSel2,(LPSTR)"")!=0)
            {
        EnableMenuItem(GetMenu(hWnd),ID_DESCRENT2,MF_ENABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_TREE2,MF_ENABLED|MF_BYCOMMAND);
             Tree2=InitTree(hhh,File2,EntSel2);
             LAtt2=ConvertTree(hhh,Tree2,LAtt2,TRUE);
             LA2Save;
            };
        if((lstrcmp(EntSel1,(LPSTR)"")!=0)&&(lstrcmp(EntSel2,(LPSTR)"")!=0))
            {
        EnableMenuItem(GetMenu(hWnd),ID_SELECTATTR,MF_ENABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_CORRENT,MF_ENABLED|MF_BYCOMMAND);
            AutoAid(hhh,LAtt1,LAtt2,"AA");
            };
         ProcessParam(hhh);
        break;

case ID_SELECTATTR:
        EnableMenuItem(GetMenu(hWnd),ID_DESCRATTR1,MF_DISABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_DESCRATTR2,MF_DISABLED|MF_BYCOMMAND);
        EnableMenuItem(GetMenu(hWnd),ID_CORRATT,MF_DISABLED|MF_BYCOMMAND);
        lstrcpy((LPSTR)AttSel1,(LPSTR)"");
        lstrcpy((LPSTR)AttSel2,(LPSTR)"");
        AttSel(hhh);
        ProcessParam(hhh);
        if(lstrcmp((LPSTR)AttSel1,(LPSTR)"")!=0)
        EnableMenuItem(GetMenu(hWnd),ID_DESCRATTR1,MF_ENABLED|MF_BYCOMMAND);
        if(lstrcmp((LPSTR)AttSel2,(LPSTR)"")!=0)
        EnableMenuItem(GetMenu(hWnd),ID_DESCRATTR2,MF_ENABLED|MF_BYCOMMAND);
        if((lstrcmp((LPSTR)AttSel1,(LPSTR)"")!=0)&&
          (lstrcmp((LPSTR)AttSel2,(LPSTR)"")!=0))
        EnableMenuItem(GetMenu(hWnd),ID_CORRATT,MF_ENABLED|MF_BYCOMMAND);
       break;

case ID_DESCRSCH1:
          fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_CREATE);
          InitDescrSchema(hWnd,File1);
          ProcessDescr(hWnd);
          fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_DELETE);
```

```
        break;

case ID_DESCRSCH2:
        fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_CREATE);
        InitDescrSchema(hWnd,File2);
        ProcessDescr(hWnd);
        fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_DELETE);
        break;

case ID_DESCRENT1:
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_CREATE);
         InitDescrEnt(hWnd,File1,EntSel1);
         ProcessDescr(hWnd);
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_DELETE);
       break;

case ID_DESCRENT2:
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_CREATE);
         InitDescrEnt(hWnd,File2,EntSel2);
         ProcessDescr(hWnd);
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_DELETE);
       break;

case ID_DESCRATTR1:
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_CREATE);
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_DELETE);

       break;

case ID_DESCRATTR2:
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_CREATE);
         fromf = OpenFile("toto",(LPOFSTRUCT)&ofs1,OF_DELETE);
       break;

case ID_TREE1:
         ProcessTree1(hhhh);
       break;

case ID_TREE2:
         ProcessTree2(hhhh);
       break;

case ID_CORRENT:
     lpa = MakeProcInstance(correntproc, hInst);
     DialogBox (hInst, (LPSTR)"CORRENT", hhh, lpa);
     FreeProcInstance((FARPROC)lpa);
     break;

case ID_CORRATT:
     lpa = MakeProcInstance(corrattproc, hInst);
     DialogBox (hInst, (LPSTR)"CORRATT", hhh, lpa);
     FreeProcInstance((FARPROC)lpa);
     break;

default:
     return (DefWindowProc(hWnd, message, wId, lParam));
     break;

};
```

```
return (0);
}

/*****************************************************************************/
```

```
/*****************************************************************************/
/*                                  READCAR                                  */
/*****************************************************************************/
/* Retourne le caractere se trouvant a la position offs dans le fichier
f. Entre deux appels a cette procedure, celle-ci memorise le block courant
dans rdoffs qui est static. Ceci evite de relire a chaque fois le fichier
pour lire un seul caractere. Le bloc est change si le fichier est different
de celui de l'appel precedent ou sil l'offset demande se trouve a l'exterieur
du block memorise */

int readcar(f, offs)

char *f;
long  offs;

{
static long rdoffs;

if ((strcmp(f, factif) != 0) || (offs < stoffs) || (offs > (stoffs+LBLOCK-1)))
    {
      rdoffs= readblock(f, offs, (LPSTR) stbuf);
      if (rdoffs < 0) return (-1);
      lstrcpy((LPSTR)factif, (LPSTR)f);
      stoffs=offs;
    }
if (offs > rdoffs) return(-1);
return (stbuf[offs-stoffs]);
}

/*****************************************************************************/
/*                              FINDOBJECT                                   */
/*****************************************************************************/
/* Cherche dans le fichier f, a partir de la position offset s'il trouve
le string contenu dans obj. Si oui, il renvoie l'offset du caractere suivant
le dernier caractere du string dans le fichier; sinon il renvoie -1 */

long findobject (f, offset, obj)

char *f;
long offset;
char *obj;

{
long myoffs;
int i;
int current;


myoffs= offset;
i=0;
while (TRUE)
    {
     if (obj[i] == '\0') return(myoffs);
     else {
           current = readcar(f, myoffs);
           if (current < 0) return (-1);
           else if (current == obj[i]) i++;
                else i=0;
           myoffs++;
```

```
        }
    }

}

/*****************************************************************************/
/*                              FINDNAME                                     */
/*****************************************************************************/
/* Lit dans le fichier f, a partir de la position offset et renvoie ce qu'il a
lu dans le string name. La lecture s'arrete au premier blanc, ; ou fin de
fichier. Renvoie -1 si probleme */

long FAR PASCAL findname(f, offset, name)

char *f;
long offset;
char *name;

{
BOOL fini = FALSE;
int current;
long myoffs;

myoffs= offset;
while (! fini)
      {
       current= readcar(f, myoffs);
       if (current<0) return (-1);
       else myoffs++;
       if ((current == ' ') || (current==''))||(current==''))
       ;
       else
            {
             name[0]=current;
             name++;
             fini = TRUE;
            }
      }
fini = FALSE;
while (! fini)
 {
  current= readcar(f, myoffs);
  if (current<0) return (-1);
  if ((current == ' ') || (current == ';') || (current == '')
     || (current == ''))
  {name[0]='\0';fini= TRUE;}
   else { name[0]=current;
          name++;
          myoffs++;
        }
  }
return(myoffs);
}
```

```
/*****************************************************************************/
/*                        R E A D B L O C K                                  */
/*****************************************************************************/
/*Lit dans le fichier fd un bloc de taille LBLOCK (ou moins si fin de fichier)
a partir de la position offset et le place dans la zone pointee par buf.
La fonction renvoie la position courante dans le fichier si la lecture s'est
bien deroulee, -1 sinon.*/

long readblock (fd, offset, buf)

char *fd;
long offset;
LPSTR buf;


{
OFSTRUCT ofstruct;
int ftext;
char bbuf[LBLOCK+1];
int l;
long ll;


if (OpenFile((LPSTR) fd, (LPOFSTRUCT) &ofstruct, OF_PARSE) == -1)
{
return(-1);
}
if ((ftext=open(fd, O_BINARY)) == -1)
{
return(-1) ;
}
lseek(ftext,(LONG) offset, 0);

if (( l = read(ftext, bbuf, LBLOCK)) == -1)
{
return(-1)  ;
}

bbuf[l] = '\0';
lstrcpy(buf, (LPSTR) "\0");
lstrcat((LPSTR) buf, (LPSTR) bbuf);

ll = (long) offset + (long) l;

close(ftext);

return (ll);
}

/*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                        M O D U L E R E A D S D C                          */
/*                                                                           */
/*****************************************************************************/
/* Ce module comporte un ensemble de fonctions travaillant sur un fichier
repondant a la syntaxe SDC */


/*****************************************************************************

Manipulation du fichier 'fcorr'.
exemple:                    { '_' = caractere(s) blanc(s)!!!!!! }

MS_schema1_schema2;         { lignes terminees par ';' }
CS_schema1_schema2;
*_EE_nom1_*_id_*_nom2;      { '*' au debut = proposition de corr }
EE_nom3_*_*_nom4;           { '*__*' = corr non precise }
CS_schema2_schema1;
*_EE_nom2_*_id_*_nom1;
EE_nom4_*_*_nom3;
                            { !!!! le fichier se termine par un '.' !!!!! }
types codes sur 2 lettres:
identite.............id
plus  precis.........pp
moins  precis........mp
fonction.............f+
fonction inverse.....f-
angle de vue.........av
sous-type  de........st
st/st................ss
non  precise.........__



*****************************************************************************/



/*****************************************************************/
/*                          LCALLOC                              */
/*****************************************************************/
/* Alloue la place necessaire a un element de type lcorrelem */

HANDLE FAR PASCAL lcalloc(hwnd)

HWND hwnd;
{
int longueur;
HANDLE hGlob;

longueur = sizeof(struct lcorrelem);
hGlob = GlobalAlloc(GHND,(long)longueur);
if (hGlob == NULL) MessageBox(hwnd,(LPSTR)"ALLOC RATEE",NULL,MB_OK);
return (hGlob);

}
```

```
/*********************************************************************/
/*                              ALC                                  */
/*********************************************************************/
/* Ajoute une correspondance a la liste ll, les champs sont rassembles
dans le string e. A la difference d'AELC (ci-dessous) elle ne s'occupe
pas des correspondances se trouvant deja dans ll. Cette procedure est
utilisee pour le chargement lors du changement des schemas de travail */

LCORR alc(e, ll, hwnd)

char *e;
LCORR ll;
HWND hwnd;
{
LCORR l, nouv;
HANDLE h;
char buf[50];
int j;

h=lcalloc(hwnd);
nouv=(LCORR)GlobalLock(h);

if  (*e=='C')
{
lstrcpy((LPSTR)nouv->dauto ,(LPSTR)"" );
j=0;
lstrcpy((LPSTR)buf,(LPSTR)"");
for (e=e+3; *e != ' '; buf[j++]=*e++);
buf[j]='\0';
e++;
lstrcpy((LPSTR)nouv->s1 ,(LPSTR)buf );
j=0;
lstrcpy((LPSTR)buf,(LPSTR)"");
for (; *e != '\0'; buf[j++]=*e++);
buf[j]='\0';
lstrcpy((LPSTR)nouv->s2 ,(LPSTR)buf );
lstrcpy((LPSTR)nouv->obj ,(LPSTR)"CS" );
lstrcpy((LPSTR)nouv->type ,(LPSTR)"" );
lstrcpy((LPSTR)nouv->n1 ,(LPSTR)"" );
lstrcpy((LPSTR)nouv->n2 ,(LPSTR)"" );
}
else
   {
    if  (*e=='*')
       {
       lstrcpy((LPSTR)nouv->dauto,(LPSTR)"*");
        e=e+2;
       }
    else lstrcpy((LPSTR)nouv->dauto,(LPSTR)"");
    j=0; lstrcpy((LPSTR)buf,(LPSTR)"");
    for (; *e != ' '; buf[j++]=*e++);
    buf[j]='\0';
    e++;
    lstrcpy((LPSTR)nouv->obj ,(LPSTR)buf );
    j=0; lstrcpy((LPSTR)buf,(LPSTR)"");
    for (; *e != ' '; buf[j++]=*e++);
    buf[j]='\0';
    e++;
```

```
    lstrcpy((LPSTR)nouv->n1 ,(LPSTR)buf );
    e=e+2;
    if (*e == '*')
       {
        lstrcpy((LPSTR)nouv->type,(LPSTR)"");
        e=e+2;
       }
    else
        {
        j=0; lstrcpy((LPSTR)buf,(LPSTR)"");
        for (; *e != ' '; buf[j++]=*e++);
        buf[j]='\0';
        e=e+3;
        lstrcpy((LPSTR)nouv->type ,(LPSTR)buf );
        }
    j=0; lstrcpy((LPSTR)buf,(LPSTR)"");
    for (; *e != '\0'; buf[j++]=*e++);
    buf[j]='\0';
    lstrcpy((LPSTR)nouv->n2 ,(LPSTR)buf );
    }
nouv->next=NULL;
if (ll==NULL) {return(nouv);}
l=ll;
while (l->next != NULL)
       l=l->next;
l->next=nouv;
return(ll);
}

/*************************************************************************/
/*                                CLC                                    */
/*************************************************************************/
/* Construit a partir du fichier fcorr, la liste( de type LCORR)
 des correspondances entre les elements des schemas s1 et s2 */

LCORR clc(s1, s2)

char *s1, *s2;

{
LCORR lc=NULL;

LONG offs;
char buf[LBLOCK];
char c;
int j;

strcpy(buf, "");
strcat(buf, "MS");
strcat(buf," " );
strcat(buf,s1 );
strcat(buf," " );
strcat(buf,s2 );
strcat(buf,";" );
offs= findobject("fcorr",(LONG)0, buf);
if (offs < (LONG)0)
   {
    strcpy(buf, "");
    strcat(buf, "MS");
```

```
    strcat(buf," " );
    strcat(buf,s2 );
    strcat(buf," " );
    strcat(buf,s1 );
    strcat(buf,";" );
    offs= findobject("fcorr",(LONG)0, buf);
    if ( offs <= (LONG)0)
       MessageBox(hhh, (LPSTR)"fcorr not conform", NULL, MB_OK);
  }

c = readcar("fcorr", offs++);
if(c == '') {c=readcar("fcorr", offs++);c=readcar("fcorr", offs);};
while ((c != 'M')&&(c != '.'))
{
 j=0;
 while (c != ';')
     {
      buf[j++]=c;
      c=readcar("fcorr", offs++);
      if(c == '') {c=readcar("fcorr", offs++);c=readcar("fcorr", offs++);};
     }
 buf[j]='\0';
 lc = alc(buf,lc,hhh);
 c=readcar("fcorr", offs++);
if(c == '') {c=readcar("fcorr", offs++);c=readcar("fcorr", offs++);};
}
return(lc);
}

/*********************************************************************/
/*                              AELC                                 */
/*********************************************************************/
/* Ajoute un element a la liste (de type FCORR) lc, tous les champs de
ce nouvel element se trouvent dans les autres parametres. Cette procedure
teste les anciens elements de lc de maniere a eviter que la nouvelle
correspondance n'apporte une redondance dans lc.  */

LCORR aelc( lc, dauto, s1, s2, obj, type, n1, n2)

LCORR lc;
char *dauto, *s1, *s2, *obj, *type;
LPSTR n1,n2;

{
LCORR ll;
HANDLE h;
LCORR nouv, prec;

ll=lc;
h=lcalloc(hhh);
nouv=(LCORR)GlobalLock(h);
lstrcpy((LPSTR)nouv->dauto ,(LPSTR)dauto );
lstrcpy((LPSTR)nouv->s1 ,(LPSTR)s1 );
lstrcpy((LPSTR)nouv->s2 ,(LPSTR)s2 );
lstrcpy((LPSTR)nouv->obj ,(LPSTR)obj );
lstrcpy((LPSTR)nouv->type ,(LPSTR)type );
lstrcpy((LPSTR)nouv->n1 ,(LPSTR)n1 );
lstrcpy((LPSTR)nouv->n2 ,(LPSTR)n2 );
```

```
if ( lstrcmp((LPSTR)(ll->s1),(LPSTR)(nouv->s1)) == 0)
{
  prec = ll;
  ll=  ll->next;
}
else
{
  prec=ll;
  ll=ll->next;

  while (lstrcmp((LPSTR)(ll->obj),(LPSTR)"CS") != 0)
   {
    ll=ll->next;
   };
  prec = ll;
  ll=ll->next;
}
while ((lstrcmp((LPSTR)ll->n1,(LPSTR) n1) !=0) && (ll != NULL) &&
                  (lstrcmp((LPSTR)ll->obj,(LPSTR)"CS") != 0))
      {
        prec = ll;
        ll=ll->next;
      }
if (lstrcmp((LPSTR)ll->n1,(LPSTR)n1) ==0)
{
if(
   ((strcmp(dauto,"")==0)&&(strcmp(type,"")!=0))  ||
   ((strcmp(type,  "")!=0)&&(lstrcmp((LPSTR)ll->type,(LPSTR)"")==0))||
   (((strcmp(dauto,  "")==0)&&(lstrcmp((LPSTR)ll->dauto,(LPSTR)"")!=0))&&
     (lstrcmp((LPSTR)ll->type,(LPSTR)"")==0))
  )
   { nouv->next = ll->next;
     prec->next = nouv;
     return(lc);
   }
}

if (ll == NULL)
{
  prec->next = nouv;
  nouv->next = NULL;
  return(lc);
}

if (lstrcmp((LPSTR)ll->obj,(LPSTR)"CS") == 0)
{

nouv->next = ll;
prec->next = nouv;
return(lc);
}
return (lc);
}

/*********************************************************************/
/*                              WLC                                  */
/*********************************************************************/
/* Remplace la liste des correspondances entre les deux schemas courants
se trouvant dans fcorr par la nouvelle liste (l'ancienne plus eventuellement
```

```
de nouvelles correspondances)construite par le programme.  */

BOOL  wlc(lc)

LCORR lc;

{
OFSTRUCT ofs1, ofs2;
int fromf, tof,res;
int l;
char bbuf[LBLOCK + 1];
LONG cur = 0; /* dernier byte qui vient d'etre copie */
LPSTR b;
int i;

LONG offset, noffs;
char obj[240];
LCORR ll;

lstrcpy((LPSTR)obj,(LPSTR) "");
lstrcat((LPSTR)obj,(LPSTR) "MS ");
lstrcat((LPSTR)obj,(LPSTR)(lc->s1));
lstrcat((LPSTR)obj,(LPSTR) " ");
lstrcat((LPSTR)obj,(LPSTR)lc->s2);
lstrcat((LPSTR)obj,  (LPSTR)";");

offset= findobject("fcorr", (LONG)0, obj)  +(LONG)1;

/* debut bloc a remplacer */
noffs= findobject("fcorr", (LONG)offset, ";MS");
noffs = noffs-(LONG)1;

/* fin bloc à remplacer */

if ((fromf = OpenFile("fcorr",(LPOFSTRUCT)&ofs1, OF_READ)) == -1)
  { MessageBox(hhh,(LPSTR)"probleme avec fromf",NULL,MB_OK);
    return(FALSE);};
if ((tof = OpenFile("inter",(LPOFSTRUCT)&ofs2,OF_CREATE|OF_WRITE)) == -1)
  { MessageBox(hhh,(LPSTR)"probleme avec tof",NULL,MB_OK);
    return(FALSE);};


while ((offset - cur) > (LONG) LBLOCK)
    {
    if ((l = read(fromf, bbuf, LBLOCK)) < LBLOCK)
     { MessageBox(hhh,(LPSTR)"probleme avec read fromf 1",NULL,MB_OK);
       return(FALSE);};
    if ((l = write(tof, bbuf, LBLOCK)) < LBLOCK)
     { MessageBox(hhh,(LPSTR)"probleme avec write tof 1",NULL,MB_OK);
       return(FALSE);};
    cur += (LONG) l;
    }

if (( offset - cur ) > (LONG) 1)
   {
    res=(int)(offset-cur);
    if ((l = read(fromf, bbuf,(int) (offset - cur -(LONG) 1)))
                                  < (offset - cur - (LONG)1))
     { MessageBox(hhh,(LPSTR)"probleme avec read fromf 2",NULL,MB_OK);
```

```
      return(FALSE);};              /* probleme */

    if ((l = write(tof, bbuf,(int) (offset - cur - (LONG)1)))
                                   < (offset - cur - (LONG)1))
     { MessageBox(hhh,(LPSTR)"probleme avec write tof 2",NULL,MB_OK);
       return(FALSE);};
    cur += (LONG) l;
  }
/* ici offset == cur+1 */

/* insert new list here */
ll=lc;
while (ll != NULL)
{
strcpy(bbuf, "");
if(lstrcmp((LPSTR)ll->dauto,(LPSTR)"")!=0) strcat(bbuf, "* ");
lstrcat((LPSTR)bbuf,(LPSTR) ll->obj); strcat(bbuf, " ");
if (lstrcmp((LPSTR)ll->obj,(LPSTR) "CS")==0)
    {
     lstrcat((LPSTR)bbuf,(LPSTR)ll->s1 );
     strcat(bbuf," " );
     lstrcat((LPSTR)bbuf,(LPSTR)ll->s2 );
    }
else
    {
     lstrcat((LPSTR)bbuf,(LPSTR)ll->n1 );
     strcat(bbuf," * " );
     if (lstrcmp((LPSTR)ll->type,(LPSTR) "")!=0)
        {
          lstrcat((LPSTR)bbuf,(LPSTR)ll->type );
          strcat(bbuf," " );
        }
     strcat(bbuf, "* ");
     lstrcat((LPSTR)bbuf,(LPSTR) ll->n2);
    }
strcat(bbuf, ";");
l= write(tof, bbuf, strlen(bbuf));
ll=ll->next;
}

if (noffs >= (LONG)0)
{
      /* skip old list */
       offset=noffs;

        while ((offset - cur) > (LONG) LBLOCK)
            {
             if ((l = read(fromf, bbuf, LBLOCK)) < LBLOCK)
              { MessageBox(hhh,(LPSTR)"probleme avec read fromf 1",NULL,MB_OK);
                return(FALSE);};
                cur = cur + (LONG) l;
            }

        if (( offset - cur ) > (LONG) 1)
          {
           res=(int)(offset-cur);
           if ((l = read(fromf, bbuf,(int) (offset - cur -(LONG) 1)))
                                   < (offset - cur - (LONG)1))
             { MessageBox(hhh,(LPSTR)"probleme avec read fromf 2",NULL,MB_OK);
```

```
                return(FALSE);};            /* probleme */

          }

  /* copy what remains */


  l = LBLOCK;
  while (l != 0)
    {
    if ((l = read(fromf, bbuf, LBLOCK)) == -1)
     { MessageBox(hhh,(LPSTR)"probleme avec read fromf 3",NULL,MB_OK);
       return(FALSE);};
    if (l != 0) /* pas fin de fichier */
       {if ((l = write(tof, bbuf, l)) == -1)
             { MessageBox(hhh,(LPSTR)"probleme avec write tof 3",NULL,MB_OK);
               return(FALSE);};
       };
    };
}

else {l= write(tof, ".", 1);
     }


close (fromf);
close (tof);

fcopy (hhh, "inter", "fcorr");
 OpenFile ((LPSTR) "inter", (LPOFSTRUCT) &ofs2, OF_DELETE) == -1;

return(TRUE);
}



/*********************************************************************/
/*                              WFIRSTLC                             */
/*********************************************************************/
/* Initialise fcorr: remplit fcorr par lc en partant du principe que
fcorr n'existe pas encore */

BOOL wfirstlc(lc)

LCORR lc;

{
OFSTRUCT ofs1, ofs2;
int fromf, tof,res;
int l;
char bbuf[LBLOCK + 1];
LONG cur = 0; /* dernier byte qui vient d'etre copie */
LPSTR b;
int i;

LONG offset, noffs;
char obj[50];
LCORR ll;
```

```
if ((tof = OpenFile("fcorr",(LPOFSTRUCT)&ofs1,OF_CREATE|OF_WRITE)) == -1)
  { MessageBox(hhh,(LPSTR)"probleme avec fromf",NULL,MB_OK);
    return(FALSE);};

/* insert new list here */

ll=lc;
while (ll != NULL)
{
strcpy(bbuf, "");
if(lstrcmp((LPSTR)ll->dauto,(LPSTR)"")!=0) strcat(bbuf, "* ");
lstrcat((LPSTR)bbuf,(LPSTR) ll->obj); strcat(bbuf, " ");
if  ((lstrcmp((LPSTR)ll->obj,(LPSTR)  "CS")==0)||
    (lstrcmp((LPSTR)ll->obj,(LPSTR)  "MS")==0))
    {
     lstrcat((LPSTR)bbuf,(LPSTR)ll->s1 );
     strcat(bbuf," " );
     lstrcat((LPSTR)bbuf,(LPSTR)ll->s2 );
    }
else
    {
     lstrcat((LPSTR)bbuf,(LPSTR)ll->n1 );
     strcat(bbuf," * " );
     if (lstrcmp((LPSTR)ll->type,(LPSTR) "")!=0)
        {
          lstrcat((LPSTR)bbuf,(LPSTR)ll->type );
          strcat(bbuf," " );
        }
     strcat(bbuf, "* ");
     lstrcat((LPSTR)bbuf,(LPSTR) ll->n2);
    }
strcat(bbuf, ";");
l= write(tof, bbuf, strlen(bbuf));
ll=ll->next;
}

l= write(tof, ".", 1);

close (tof);

return(TRUE);
}

/*****************************************************************/
/*                          INITSDC                              */
/*****************************************************************/
/* Initialise fcorr en construisant une liste de "correspondances"
minimales (MS,CS) qu'il fournira a wfirstlc */

FAR PASCAL InitSdc(hwnd)

HWND hwnd;

{
 LISTESCH noeud1,noeud2;
 HANDLE h;
 LCORR nouv1,nouv2,nouv3;
```

```
ListCorr = NULL;
noeud1=listsch;
while(noeud1!=NULL)
  {
   noeud2=noeud1->next;
   while(noeud2!=NULL)
     {

      h=lcalloc(hwnd);
      nouv1=(LCORR)GlobalLock(h);
      if (ListCorr==NULL){ListCorr=nouv1;}
      else { nouv3->next=nouv1;};
      lstrcpy((LPSTR)nouv1->s1,(LPSTR)noeud1->schn);
      lstrcpy((LPSTR)nouv1->s2,(LPSTR)noeud2->schn);
      lstrcpy((LPSTR)nouv1->obj,(LPSTR)"MS");

      h=lcalloc(hwnd);
      nouv2=(LCORR)GlobalLock(h);
      nouv1->next=nouv2;
      lstrcpy((LPSTR)nouv2->s1,(LPSTR)noeud1->schn);
      lstrcpy((LPSTR)nouv2->s2,(LPSTR)noeud2->schn);
      lstrcpy((LPSTR)nouv2->obj,(LPSTR)"CS");

      h=lcalloc(hwnd);
      nouv3=(LCORR)GlobalLock(h);
      nouv2->next=nouv3;
      lstrcpy((LPSTR)nouv3->s1,(LPSTR)noeud2->schn);
      lstrcpy((LPSTR)nouv3->s2,(LPSTR)noeud1->schn);
      lstrcpy((LPSTR)nouv3->obj,(LPSTR)"CS");

      noeud2=noeud2->next;
     };

   noeud1=noeud1->next;
  };

      /*        Imprimer dans le fichier */
wfirstlc(ListCorr);

}

/*****************************************************************/
```

```
/*************************************************************************/
/*                                                                       */
/*                    M O D U L E L I S T A T T                          */
/*                                                                       */
/*************************************************************************/
/* Gere la creation de la liste, sous forme arborescente, des attributs
d'une entite */


/*************************************************************************/
/*                              FINDPOSTDEF                              */
/*************************************************************************/
/* Lit dans le fichier f, a partir de la position offset. Remplit le string
name avec se qu'il lit, jusqu'a ce qu'il trouve un ; */

long FAR PASCAL FindPostDef(f, offset, name)

char *f;
long offset;
char *name;

{
BOOL fini = FALSE;
int current;
long myoffs;

myoffs= offset;
fini = FALSE;
while (! fini)
      {
       current= readcar(f, myoffs);
       if (current<0) return (-1);
       if (current == ';')
       {name[0]=';';name++;name[0]='\0';fini=  TRUE;}
       else
           {
            if (current == '') {myoffs=myoffs+2;}
            else
            { name[0]=current;
              name++;
              myoffs++;
            }
           }
      }
return(myoffs);
}

/*************************************************************************/
/*                              CUTLIST                                  */
/*************************************************************************/
/* Une liste d'attribut en DSL (debute par CONSISTS OF) est composee
d'un ensemble de nom separe par des , et terminee par un ;. La procedure
FindPostDef renvoie la liste entiere. CutList separe le premier element
des autres et le renvoie.  */

PSTR FAR PASCAL CutList(strs)

char strs[];
```

```
{
 PSTR petitstr;
 int  i=0,j=0;

 strcpy(petitstr,";");
 while((strs[i]==' ')||(strs[i]==','))
  {
   strs[i]=' ';
   i++;
  };
 if(strs[i]==';') return(petitstr);
 petitstr[j]=strs[i];
 while((strs[i]!=' ')&&(strs[i]!=',')&&(strs[i]!=';'))
  {
    petitstr[j]=strs[i];
    strs[i]=' ';
    i++;
    j++;
  };
 petitstr[j]='\0';
 return(petitstr);
}

/*****************************************************************/
/*                          TREEALLOC                            */
/*****************************************************************/
/* Alloue la place necessaire a une structure de type elem */

HANDLE FAR PASCAL TreeAlloc(hwnd)

HWND hwnd;

{
int longueur;
int i;
HANDLE hGlob;

longueur = sizeof(struct elem);
hGlob = GlobalAlloc(GHND,(long)longueur);
if (hGlob == NULL) MessageBox(hwnd,(LPSTR)"ALLOC RATEE",NULL,MB_OK);
return (hGlob);

}

/*****************************************************************/
/*                          INITARBRE                            */
/*****************************************************************/

/* Cette procedure construit l'arbre des attributs d'une entite
(ou d'une relation)cfr intdata.c pour connaitre la structure de cet arbre
de pointeurs.
Elle doit connaitre l'offset suivant le dernier char de la denomination de
l'objet concerné ainsi que le nom du fichier contenant cet objet.
Elle renvoie un pointeur vers le debut de la structure produite. Ce pointeur
est egal a NULL si l'entite ne possede pas d'attributs.  */

InitArbre(hwnd,fd,offset,noeud)

HWND hwnd;
```

```
PSTR fd;
long offset;
LPELEM noeud;

{
 HANDLE hfils1,hfils2;
 LPELEM fils1=NULL,fils2=NULL;
 char name[200];
 char strs[200];
 char petitstr[200];
 long  noffset,offset1,offset2,offset3;

 offset = findname(fd,offset,name);
 lstrcpy(noeud->name,name);

 lstrcpy(noeud->ConMin,(LPSTR)"00");          /* connectivite a faire */
 lstrcpy(noeud->ConMax,(LPSTR)"00");

 offset1=findobject(fd,offset,"CONSISTS OF");
 offset2=findobject(fd,offset,"DEFINE");
 if(((offset1<offset2)||(offset2<=0))&&(offset1>0))
   /* decomposable */
  {
   hfils2=TreeAlloc(hwnd);
   fils2=GlobalLock(hfils2);
   fils2=NULL;
   offset3=FindPostDef(fd,offset1,strs);
   strcpy(petitstr,CutList(strs));

   while(strcmp(petitstr,";")!=0)
    {
     noffset=GroupElem(fd,petitstr,offset3);

     hfils1=TreeAlloc(hwnd);
     fils1=GlobalLock(hfils1);
     InitArbre(hwnd,fd,noffset,fils1);
     fils1->brother=fils2;
     fils2=fils1;
     strcpy(petitstr,CutList(strs));
    };
    noeud->son=fils2;
  }
 else
  /* simple */
 {
  noeud->son=NULL;
 }
}

/*************************************************************************/
/*                                INITREE                                */
/*************************************************************************/
/* Lance InitArbre,car cas special de l'entie. Premier appel different des
autres */

LPELEM InitTree(hwnd,file,name)

HWND hwnd;
char *file;
```

```
char *name;

{
 LPELEM noeud;
 HANDLE hnoeud;
 long  offset;
 PSTR fd;


 hnoeud=TreeAlloc(hwnd);
 noeud=GlobalLock(hnoeud);
 noeud->brother=NULL;
 offset  =  PosEntity(file,name,(long)0);
 InitArbre(hwnd,file,offset,noeud);
 return (noeud);
}

/*************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                        M O D U L E T R E E                                */
/*                                                                           */
/*****************************************************************************/

/* Affichage et gestion des fenΩtres contenant les arbres des attributs des
deux entites selectionnees */


/*****************************************************************************/
/*            T H E P R O C E S S T R E E P R O C E D U R E(S)               */
/*****************************************************************************/
/* Creation de la fenΩtre dans laquelle l'arbre de la premiere (ou la seconde)
apparaitra, les WS_ precisent que la fenΩtre possedera deux scroll-bars.
Certaines procedures sont dedoublees pour faciliter l'affichage simultane des
deux listes d'attributs en memoire. Si nous n'utilisons pas ce moyen, les deux
fenΩtres affichent le mΩme dessin aprÇs l'utilisation d'un scroll */

FAR PASCAL ProcessTree1(hWnd)

HWND hWnd;

{


GetClientRect(hWnd,(LPRECT)&TilWnd);
hC3Wnd = CreateWindow((LPSTR) "TREE1" , /* Window class name   */
                  (LPSTR) "TREE 1" , /* Window title           */
                  WS_CHILD|WS_SIZEBOX|WS_VISIBLE|WS_BORDER|WS_CAPTION
                  |WS_SYSMENU|WS_VSCROLL|WS_HSCROLL|WS_CLIPSIBLINGS,
                  0                , /* x - ignored for tiled */
                  0                , /* y - ignored for tiled */
                  TilWnd.right     , /*  cx  */
                  TilWnd.bottom *2/3 , /* cy */
                  (HWND)hWnd       , /* Parent of this wind */
                  (HMENU)1         , /* Use the class menu     */
                  (HANDLE) hInst , /* Who created this window */
                  (LPSTR)NULL          /* No params. to pass on.  */
                 );
        return (1);
}


/*****************************************************************************/

FAR PASCAL ProcessTree2(hWnd)

HWND hWnd;

{


GetClientRect(hWnd,(LPRECT)&TilWnd);
hC3Wnd = CreateWindow((LPSTR) "TREE2" , /* Window class name   */
                  (LPSTR) "TREE_2" , /* Window title           */
                  WS_CHILD|WS_SIZEBOX|WS_VISIBLE|WS_SIZEBOX|WS_BORDER
                  |WS_CAPTION|WS_SYSMENU|WS_VSCROLL|WS_HSCROLL
                  |WS_CLIPSIBLINGS,
```

```
                    10                , /* x - ignored for tiled */
                    10                , /* y - ignored for tiled */
                    TilWnd.right      , /*  cx  */
                    TilWnd.bottom *2/3 , /* cy */
                    (HWND)hWnd        , /* Parent of this wind */
                    (HMENU)1          , /* Use the class menu     */
                    (HANDLE) hInst , /* Who created this window */
                    (LPSTR)NULL          /* No params. to pass on.  */
                   );
          return (1);
}


/*************************************************************************/
/*          T H E T R E E W I N D O W P R O C E D U R E(S)        */
/*************************************************************************/
/* Gestion des messages envoyes a la fenetre TREE, ici il faut tenir compte de
la prθsence des scrolls-bars. Cette procedure est appelee par WINDOWS */

LONG FAR PASCAL TreeWindowProc1 (hWnd,
                                 message,
                                 wParam,
                                 lParam)

HWND     hWnd  ;
unsigned message;
WORD     wParam ;
LONG     lParam ;

{
        switch (message)
        {
                case WM_CREATE :
                                break;

                case WM_VSCROLL :
                                VScrolling (hWnd,wParam,lParam);
                                break;

                case WM_HSCROLL :
                                 HScrolling (hWnd, wParam, lParam);
                                 break;

                case WM_PAINT :
                                PaintTreeWindow (hWnd,Tree1);
                                break;

                default :
                                return ( DefWindowProc(hWnd, message, wParam,
                                         lParam));
                                break;
        }
        return (0L);
}

/*************************************************************************/

LONG FAR PASCAL TreeWindowProc2 (hWnd,
                                 message,
```

```
                                        wParam,
                                        lParam)

HWND     hWnd  ;
unsigned message;
WORD     wParam ;
LONG     lParam ;

{
        switch (message)
        {
                case WM_CREATE :
                                break;

                case WM_VSCROLL :
                                VScrolling (hWnd,wParam,lParam);
                                break;

                case WM_HSCROLL :
                                 HScrolling (hWnd, wParam, lParam);
                                 break;

                case WM_PAINT :
                                PaintTreeWindow (hWnd,Tree2);
                                break;

                default :
                                return ( DefWindowProc(hWnd, message, wParam,
                                        lParam));
                                break;
        }
        return (0L);
}

/*************************************************************************/
/* P R O C E S S T H E V S C R O L L B A R                               */
/*************************************************************************/
/* Gestion du scroll vertical par gestion de la coordonnΘe en Y: TreeOrgY.
Ensuite envoi d'un InvalidateRect pour rafraichir la fenΩtre Tree concernee*/

BOOL FAR PASCAL VScrolling(hWnd, wParam, lParam)


HWND     hWnd;
WORD     wParam;
LONG     lParam;

{

BOOL bScroll;

bScroll = FALSE;

        switch (wParam)
        {
                case SB_LINEUP :
                     TreeOrgY -= Gap;
                     bScroll = TRUE;
                     break;
```

```
                case SB_LINEDOWN :
                     TreeOrgY += Gap;
                     bScroll = TRUE;
                     break;

                case SB_PAGEUP :
                     TreeOrgY -= PageH;
                     bScroll = TRUE;
                     break;

                case SB_PAGEDOWN :
                     TreeOrgY += PageH;
                     bScroll = TRUE;
                     break;

                case SB_TOP :
                     TreeOrgY = 0;
                     bScroll = TRUE;
                          break;

                case SB_BOTTOM :
                     bScroll = TRUE;
                          break;

                case SB_THUMBPOSITION :
                     TreeOrgY = (int)LOWORD(lParam);
                     bScroll = TRUE;
                          break;

 };

if  (bScroll)
  {
   TreeOrgY = max (TreeOrgY, 0);
   SetScrollPos (hWnd, SB_VERT, TreeOrgY, (BOOL) TRUE);
   InvalidateRect (hWnd, (LPRECT)NULL, (BOOL)TRUE);
  }
return TRUE;


}


/*************************************************************************/
/* P R O C E S S T H E H S C R O L L B A R                               */
/*************************************************************************/
/* Idem VScroll mais sur l'axe des X */

BOOL FAR PASCAL HScrolling(hWnd, wParam, lParam)

HWND     hWnd;
WORD     wParam;
LONG     lParam;

{

BOOL bScroll;
```

```
bScroll = FALSE;

        switch (wParam)
        {
                case SB_LINEUP :
                     TreeOrgX -= Gap;
                     bScroll = TRUE;
                     break;

                case SB_LINEDOWN :
                     TreeOrgX += Gap;
                     bScroll = TRUE;
                     break;

                case SB_PAGEUP :
                     TreeOrgX -= PageW;
                     bScroll = TRUE;
                     break;

                case SB_PAGEDOWN :
                     TreeOrgX += PageW;
                     bScroll = TRUE;
                     break;

                case SB_TOP :
                     TreeOrgX = 0;
                     bScroll = TRUE;
                           break;

                case SB_BOTTOM :
                     bScroll = TRUE;
                           break;

                case SB_THUMBPOSITION :
                     TreeOrgX = (int)LOWORD(lParam);
                     bScroll = TRUE;
                           break;
        }


if  (bScroll)
  {
   TreeOrgX = max (TreeOrgX, 0);
   SetScrollPos (hWnd, SB_HORZ, TreeOrgX, (BOOL) TRUE);
   InvalidateRect (hWnd, (LPRECT)NULL, (BOOL)TRUE);
  }
return TRUE;


}


/*****************************************************************************/
/*                              PRINTTREE                                    */
/*****************************************************************************/
/* exploration de l'arbre des attributs par rΘcursivitΘ. A chaque niveau,
calcul de la position du nom de l'attribut courant et affichage a cette
position. La position depend de son niveau et de son ordre dans la liste de ses
freres. Le premier etant le plus a gauche, tout en evitant de recouvrir les
```

```
autres noms, ecris precedemment. Pour cela une borne maximale est installee:
MaxBrother */

PrintTree(hwnd,noeud,hDC,X,Y,FATHX,FATHY)

HWND hwnd;
LPELEM noeud;

HDC hDC;
int X;
int Y;
int FATHX;
int FATHY;

{
int LongAttr;
BOOL Entite = FALSE;

if (EntiteGlob) Entite = TRUE;
EntiteGlob = FALSE;

LongAttr = lstrlen(noeud->name);

if ((noeud->son) != NULL)
      {  MoveTo(hDC,X,Y);
         LineTo(hDC,X,Y+(3*Gap));
         PrintTree(hwnd,noeud->son,hDC,X,Y+(3*Gap),X,Y);};

if ((noeud->brother) != NULL)
      {  /*  MoveTo(hDC,X+(LongAttr*FontW),Y);  DROITE  */
         MoveTo(hDC,FATHX,FATHY);              /* OBLIQUE */
         if (X+(LongAttr*FontW) > MaxBrother)
               MaxBrother = X+(LongAttr*FontW)+(5*FontW);
         LineTo(hDC,MaxBrother,Y);
         PrintTree(hwnd,noeud->brother,hDC,MaxBrother,Y,FATHX,FATHY); };

if (X >= MaxBrother) MaxBrother = X+(LongAttr*FontW)+(5*FontW);

if  (!Entite)
   {TextOut(hDC,X,Y,(LPSTR)noeud->ConMin,(WORD)2);
    TextOut(hDC,X+(2*FontW),Y,(LPSTR)"-",(WORD)1);
    TextOut(hDC,X+(3*FontW),Y,(LPSTR)noeud->ConMax,(WORD)2);
    TextOut(hDC,X,Y+Gap,(LPSTR) noeud->name,(WORD)LongAttr);}
else TextOut(hDC,X,Y-Gap,(LPSTR) noeud->name,(WORD)LongAttr);

}


/*************************************************************************/
/*          T H E P A I N T T R E E P R O C E D U R E                    */
/*************************************************************************/
/* Appelee lors de la creation et de la remise a jour de la fenΩtre TREE.
Calcule la hauteur d'une ligne en fonction du type de cractere (Font) et la
largeur d'un caractere. Appel de PrintTree avec ces nouveaux parametres */

BOOL FAR PASCAL PaintTreeWindow (hWnd,TreeP)

HWND    hWnd  ;
LPELEM  TreeP  ;
```

```
{

PAINTSTRUCT ps ;
HDC         hDC;
RECT        rWorkRect;

        TEXTMETRIC tmFontInfo;
        int         X,  Y,  i  ;


BeginPaint (hWnd, (LPPAINTSTRUCT) &ps);
hDC = ps.hdc;


GetClientRect (hWnd, (LPRECT)&rWorkRect);
SetWindowOrg (hDC, TreeOrgX, TreeOrgY);
SetViewportOrg (hDC, 0, 0);
SetWindowExt (hDC, rWorkRect.right, rWorkRect.bottom);
SetViewportExt (hDC, rWorkRect.right, rWorkRect.bottom);

        GetTextMetrics (hDC, (LPTEXTMETRIC) &tmFontInfo);
       Gap = ( tmFontInfo.tmExternalLeading +
                tmFontInfo.tmInternalLeading +
                tmFontInfo.tmHeight);
        FontW = tmFontInfo.tmAveCharWidth;
        PageH = rWorkRect.bottom - Gap;
        PageW = rWorkRect.right - Gap;

        X = 4 * tmFontInfo.tmAveCharWidth;
        Y  =  tmFontInfo.tmExternalLeading;

        EntiteGlob = TRUE;

        PrintTree(hWnd,TreeP,hDC,FontW,2*Gap,FontW,Gap);

        MaxBrother = 0;

ValidateRect (hWnd, (LPRECT)NULL);
EndPaint (hWnd, (LPPAINTSTRUCT) &ps);

return TRUE;


}

/*****************************************************************************/
```

```
/*****************************************************************/
/*                                                               */
/*                    M O D U L E R E A D D S L                  */
/*                                                               */
/*****************************************************************/
/* Ensemble de fonctions d'extraction travaillant sur des fichiers
ecrits suivant la syntaxe DSL */


/*****************************************************************/
/*                          GROUPELEM                            */
/*****************************************************************/
/* recherche l'emplacement dans le fichier fd(DSL) de la definition
(GROUP ou ELEMENT) de l'attribut se trouvant dans name. Part du principe
que le fichier est correct et donc n'arrete que quand il a trouve.
Il renvoie l'offset suivant le dernier char de GROUP ou de ELEMENT */

long GroupElem(fd,name,offset)

PSTR fd;
PSTR name;
long  offset;

{
 BOOL stop=FALSE;
 char readchr;
 long noffset;
 char newname[50];

 while(!stop)
  {
   offset=findobject(fd,offset,name);
   noffset=offset-(strlen(name)+1);
   readchr = readcar(fd,noffset);
   while(readchr==' ')
    {
     noffset--;
     readchr = readcar(fd,noffset);
    };
   if(readchr=='T')
    {
     noffset=noffset-6;
     noffset = findname(fd,noffset,newname);
     if(strcmp(newname,"ELEMENT")==0)return(noffset);
    };
   if(readchr=='P')
    {
     noffset=noffset-4;
     noffset = findname(fd,noffset,newname);
     if(strcmp(newname,"GROUP")==0)return(noffset);
    }
  }
}

/*****************************************************************/
/*                          POSENTITY                            */
/*****************************************************************/
/* recherche l'emplacement dans le fichier fd(DSL) de la definition
 de l'entite se trouvant dans name. */
```

```
long PosEntity(fd,name,offset)

PSTR fd;
PSTR name;
long offset;

{
 BOOL stop=FALSE;
 char readchr;
 long noffset;
 char newname[50];

 while(!stop)
  {
   offset=findobject(fd,offset,name);
   noffset=offset-(strlen(name)+1);
   readchr = readcar(fd,noffset);
   while(readchr==' ')
    {
     noffset--;
     readchr = readcar(fd,noffset);
    };
   if(readchr=='Y')
    {
     noffset=noffset-5;
     noffset = findname(fd,noffset,newname);
     if(strcmp(newname,"ENTITY")==0)return(noffset);
    }
  }
}

/*************************************************************************/
/*                              POSSCH                                   */
/*************************************************************************/
/* recherche et renvoie le nom du schema contenu dans le fichier fd(DSL).
Part du principeque le fichier est correct, c'est a dire qu'il contient un
schema DSL.  */

PSTR PosSch(fd)

PSTR fd;

{
 long offset;
 char name[50];

 offset=findobject(fd,(long)0,"SCHEMA");
 offset=findname(fd,offset,name);
 return((PSTR) name);
}

/*************************************************************************/
```

```
/*************************************************************************/
/*                                                                       */
/*                    M O D U L E P R O P C O R R                        */
/*                                                                       */
/*************************************************************************/
/* Rassemble les outils automatiques. Actuellement, il comprend un detecteur
de denominations identiques et un de synonymes */


/*************************************************************************/
/*           T H E A U T O A I D P R O C E D U R E                       */
/*************************************************************************/
/* Gere les outils automatiques pour detecter d'eventuelles correspondances
entre les objets des listes l1 et l2 de type LISTE */

BOOL FAR PASCAL AutoAid (hwnd,l1,l2)

HWND hwnd;
LISTE l1;
LISTE l2;

{
  IdEnt(hwnd,l1,l2);
  SynEnt(hwnd,l1,l2);
}


/*************************************************************************/
/*           D E N O M I N A T I O N I D E N T I Q U E                   */
/*************************************************************************/
/* recoit 2 listes de type LISTE les compare et retourne une liste de
propositions de correspondances entre les elements de même nom.  */

FAR PASCAL IdEnt(hwnd,l1,l2)

HWND hwnd;
LISTE l1;
LISTE l2;

{
 LISTE noeud1,noeud2;
 BOOL stop=FALSE;
 int i=1;
 char  s1[50],s2[50];

 noeud1=l1;
 noeud2=l2;
 lstrcpy((LPSTR)s1, Schema1);
 lstrcpy((LPSTR)s2, Schema2);
 while(noeud1!=NULL)
 {
  while((!stop)&&(noeud2!=NULL))
  {
   if(lstrcmp((LPSTR)(noeud1->name),(LPSTR)(noeud2->name))==0)
    {
     ListCorr= aelc(ListCorr, "*",(PSTR)s1,(PSTR)s2, "EE", "", noeud1->name,
              noeud2->name);
     stop = TRUE;
```

```
    };
   noeud2=noeud2->next;
  };
  i++;
  stop=FALSE;
  noeud2=l2;
  noeud1=noeud1->next;
 };
}

/*************************************************************************/
/*            D E N O M I N A T I O N S S Y N O N Y M E S                */
/*************************************************************************/
/* recoit 2 listes de type LISTE les compare et retourne une liste de
propositions de correspondances entre les elements de noms synonymes.  */

FAR PASCAL SynEnt(hwnd,l1,l2)

HWND hwnd;
LISTE l1;
LISTE l2;

{
 LISTE noeud1,noeud2;
 BOOL stop=FALSE;
 int i=1;
 char  s1[50],s2[50];

 noeud1=l1;
 noeud2=l2;
 lstrcpy((LPSTR)s1, Schema1);
 lstrcpy((LPSTR)s2, Schema2);
 while(noeud1!=NULL)
 {
  while((!stop)&&(noeud2!=NULL))
  {
   if(ReadSyn((LPSTR)noeud1->name,(LPSTR)noeud2->name))
    {
     ListCorr= aelc(ListCorr, "*",(PSTR)s1,(PSTR)s2, "EE", "", noeud1->name,
              noeud2->name);
     stop = TRUE;
    };
   noeud2=noeud2->next;
  };
  i++;
  stop=FALSE;
  noeud2=l2;
  noeud1=noeud1->next;
 };
}

/*************************************************************************/
/*                                READSYN                                */
/*************************************************************************/
/* Recoit deux noms et va verifier dans fsyn s'il sont synonymes,
renvoie un booleen vrai si synonymes, faux sinon */

BOOL FAR PASCAL ReadSyn(lname1,lname2)
```

```
LPSTR lname1;
LPSTR lname2;

{
 BOOL syn = FALSE;
 OFSTRUCT ofs1;
 long  offset1,offset2;
 char *ligne1;
 char *ligne2;
 int ind1,ind2;
 char name1[50];
 char name2[50];


 strcpy(name1,"");
 strcpy(name2,"");

 lstrcpy((LPSTR)name1,(LPSTR)lname1);
 lstrcpy((LPSTR)name2,(LPSTR)lname2);

 offset1=findobject("fsyn",(long)0,(PSTR)name1);
 if (offset1<1) {return(FALSE);};

 offset2=findobject("fsyn",(long)0,(PSTR)name2);
 if (offset2<1) {return(FALSE);};

 offset1=findobject("fsyn",offset1,(PSTR)"/");
 ind1=readcar("fsyn",offset1);
 offset2=findobject("fsyn",offset2,(PSTR)"/");
 ind2=readcar("fsyn",offset2);


 if(ind1==ind2){ syn = TRUE;};
 return(syn);
}

/*****************************************************************************/
```

```
/*****************************************************************/
/*                                                               */
/*              M O D U L E L I S T E N T I T Y                  */
/*                                                               */
/*****************************************************************/
/* Gere la creation d'une liste des entite d'un schema DSL */

/*****************************************************************/
/*                          LISTALLOC                            */
/*****************************************************************/
/* Alloue la place necessaire a un element de type lelem */

HANDLE listalloc(hwnd)

HWND hwnd;
{
int longueur;
HANDLE hGlob;

longueur = sizeof(struct lelem);
hGlob = GlobalAlloc(GHND,(long)longueur);
if (hGlob == NULL) MessageBox(hwnd,(LPSTR)"ALLOC RATEE",NULL,MB_OK);
return (hGlob);

}

/*****************************************************************/
/*                             ADD                               */
/*****************************************************************/
/* Ajoute un element a la liste ll de type LISTE. Le champ name de ce nouvel
element sera egal a e */

LISTE FAR PASCAL add(e, ll, hwnd)

LPSTR e;
LISTE ll;
HWND hwnd;
{
LISTE l, nouv;
HANDLE h;

h=listalloc(hwnd);
nouv=(LISTE)GlobalLock(h);
lstrcpy((LPSTR)nouv->name,(LPSTR) e);
nouv->next=NULL;

if (ll==NULL) return(nouv);

l=ll;
while (l->next != NULL)
        l=l->next;
l->next=nouv;
return(ll);
}

/*****************************************************************/
/*                          CONSLIST                             */
/*****************************************************************/
/* Construit et retourne une liste de type LISTE, constitue d'elements
```

```
de type DSL obj. Ces elements se trouvent dans le fichier ecrit dans
la syntaxe DSL fdsl */

LISTE FAR PASCAL conslist(hwnd, obj, fdsl)

HWND hwnd;
char *obj;
char *fdsl;

{
LISTE lst;
long  pos=0;
char *nom;
BOOL fin=FALSE;

lst=NULL;
while (fin == FALSE)
   {
     pos=findobject( fdsl, pos, obj);
     if (pos >= (long)0)
        {
           pos=findname( fdsl, pos, nom);
           if (pos >=(long)0) lst=add((LPSTR)nom, lst, hwnd);
           else fin = TRUE;
        }
     else fin = TRUE;
   }
return(lst);
}

/*****************************************************************************/
```

```
/*************************************************************************/
/*                    F E N E T R E T E X T E                            */
/*************************************************************************/

/* Ce fichier contient un ensemble de procedures necesaires a l'utilisation
de fenêtres textes sur WINDOWS. Nous utilisons ce type de fenêtre pour les
descriptions et le help. Nous enlevons toutes les options sauf
la scroll-bar. Ces procedures nous ont ete fournies, nous ne les detaillerons
donc pas */

/*************************************************************************/



/*************************************************************************/
/*                          TEXTENTRYPROC                                */
/*************************************************************************/

/* this is the window procedure for text-entry child window             */

LONG FAR PASCAL TextEntryProc(hwnd, message, wParam, lParam)

HWND hwnd;
unsigned message;
WORD wParam;
DWORD lParam;

{
WORD linelength = 0;


switch (message)
        {
        case WM_TEXTENTRY:
             SendMessage(hwnd, EM_LIMITTEXT, EDTEXTSIZE, 0L);
             SetScrollRange(hwnd, SB_HORZ, 0, 80, FALSE); SetScrollPos(hwnd,
             SB_HORZ, 0, TRUE); SetScrollPos(hwnd, SB_VERT, 0, TRUE); break;case
        WM_PASTE:
        case WM_KEYDOWN:
        case WM_CUT:
        case WM_COPY:
        case WM_CHAR:
             break;
        default:
        return (CallWindowProc(lpEditWndProc,hwnd, message, wParam, lParam));
        break;
        }
return(0L);
}

/*************************************************************************/
/*                          INITINSTANCE                                 */
/*************************************************************************/
/* does per-instance initialization                                      */

BOOL FAR InitInstance(hWnd,xpos,ypos,extx,exty)
```

```
HWND hWnd;
int  xpos;
int ypos;
int extx;
int exty;

{
FARPROC lpTextEntryProc;
char buf[3];


lpTextEntryProc = MakeProcInstance((FARPROC)TextEntryProc, hInst);
hTextEntry = CreateWindow(
             (LPSTR)"Edit",
             (LPSTR)NULL,
             WS_CHILD
             | WS_VSCROLL | WS_HSCROLL | WS_VISIBLE
             | ES_MULTILINE | ES_AUTOVSCROLL | ES_AUTOHSCROLL,
             xpos,
             ypos,
             extx       ,   /*   cx   */
             exty       ,   /*   cy   */
             hWnd,
             1,
             hInst,
             (LPSTR)NULL);
    if(!hTextEntry)
      {
   MessageBeep(0);
   MessageBeep(0);
   MessageBeep(0);
   MessageBeep(0);

       InitError:
       MessageBox(hEdtextWnd, (LPSTR) "Not enough memory",
(LPSTR)NULL,MB_OK|MB_ICONEXCLAMATION);
       return(FALSE);
      }
   lpEditWndProc = (FARPROC) GetWindowLong(hTextEntry,GWL_WNDPROC);
   SetWindowLong(hTextEntry, GWL_WNDPROC,(LONG)lpTextEntryProc);
   SendMessage(hTextEntry,WM_TEXTENTRY,0,0L);
}
/*************************************************************************/
/*                              BLANKTEXTBUFFER                          */
/*************************************************************************/

void BlankTextBuffer()

{
TextHandle = (LOCALHANDLE)SendMessage(hTextEntry, EM_GETHANDLE, (WORD)NULL,
             (LONG)NULL);
TextHandle = LocalFree(TextHandle);
TextHandle = LocalAlloc(LHND,(WORD)20);
SendMessage(hTextEntry, EM_SETHANDLE, (WORD)TextHandle, (LONG)NULL);
}

/*************************************************************************/
/*                              SETTEXTENTRY                             */
/*************************************************************************/
```

```
void SetTextEntry(entry)

LPSTR entry;

{
SendMessage(hTextEntry, WM_SETREDRAW, FALSE, 0L);
BlankTextBuffer();
SendMessage(hTextEntry, WM_SETREDRAW, TRUE, 0L);
SetWindowText(hTextEntry, (LPSTR)entry);
}

/*************************************************************************/
/*                              TEXTOPEN                                 */
/*************************************************************************/

BOOL FAR TextOpen (pchBuf)

char *pchBuf;

{
 int result = FALSE;
 OFSTRUCT ofStruct;
 LPSTR lpText;
 HANDLE hText;

 FILE *ftext;
 struct stat FileStat;
 char Bufin[84];

/*      Check the filename gived by the user                    */

if (OpenFile ((LPSTR)pchBuf, (LPOFSTRUCT)&ofStruct, OF_PARSE))
  {
/*  MSG_ERR(NULL,1,IDS_EDTEXT,IDS_EINVALIDFILE); ******************/
   return(FALSE);
  }
SetCursor(hWaitCurs);

/*      Read the file                                           */

if ((ftext = fopen(pchBuf,"rb")) != NULL)
  {
   stat (pchBuf,&FileStat);
   if (FileStat.st_size > EDTEXTSIZE)
     {
  /* MyMessageBox(IDS_EFILETOOBIG,pchBuf,MB_OK | MB_ICONEXCLAMATION);*/
     }
   else
     { /***         Allocate a buffer to read the text into   ** */
      hText = GlobalAlloc(GHND, (long)EDTEXTSIZE + 20);
      if (!hText)
        {
    /*  EdtextOKError(IDS_EINSMEMORY);  */

         fclose(ftext);
         return(FALSE);
        }
      lpText = GlobalLock(hText);
```

```
      lstrcpy(lpText,(LPSTR)"\0");
      count = 0;
      while(fgets(Bufin,84,ftext) != NULL)
           {
            lstrcat((LPSTR)lpText,(LPSTR)Bufin);
           }
      fclose(ftext);
      SetTextEntry(lpText);
      GlobalUnlock(hText);
      GlobalFree(hText);
      result = TRUE;
     }
   fclose(ftext);
  }
else /* if ftext == NULL                    */
    {
/*      MyMessageBox(IDS_ENOPEN,pchBuf,MB_OK |MB_ICONEXCLAMATION);     */
      result = FALSE;
    }
SetCursor (hArrowCurs);
return (result);
}

/*****************************************************************************/
```

```
/*************************************************************************/
/*                                                                       */
/*                        M O D U L E H E L P                            */
/*                                                                       */
/*************************************************************************/

/* Affichage de la fenêtre HELP */

/*************************************************************************/
/*                      P R O C E S S H E L P                            */
/*************************************************************************/
/* Procedure appelee par WINDOWS, traite les messages destines a cette
fenêtre, ici, seulement le create */

LONG FAR PASCAL HelpProc(hwnd, message, wParam, lParam)

HWND hwnd;
unsigned message;
WORD wParam;
LONG lParam;

{
RECT HelpWnd;

switch (message)
        {
        case WM_CREATE:

            GetClientRect (hwnd,(LPRECT) &HelpWnd);
            InitInstance (hwnd,0,0,HelpWnd.right,HelpWnd.bottom);

            TextOpen ("fhelp");
            break;

        default:
            return (DefWindowProc(hwnd, message, wParam, lParam));
            break;
        }
return(0L);
}

/*************************************************************************/
/*                              PROCESS HELP                             */
/*************************************************************************/
/* Creation de la fenêtre */

FAR PASCAL ProcessHelp(hWnd)

HWND hWnd;
{
RECT TilWnd;



GetClientRect(hWnd, (LPRECT) &TilWnd);
hC1Wnd = CreateWindow((LPSTR) "HELP" , /* Window class name   */
                    (LPSTR) "HELP" , /* Window title        */
                    WS_CHILD|WS_VISIBLE|WS_BORDER|WS_CAPTION|WS_SIZEBOX
                    |WS_SYSMENU|WS_CLIPCHILDREN,
```

```
                0                   , /* x - ignored for tiled */
                0                   , /* y - ignored for tiled */
                TilWnd.right        ,  /*  cx  */
                TilWnd.bottom       ,  /*  cy  */
                (HWND)hWnd          , /* Parent of this wind */
                (HMENU)1            , /* Use the class menu     */
                (HANDLE) hInst , /* Who created this window */
                (LPSTR)NULL          /* No params. to pass on.  */
               );
        return (1);

}


/*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                    M O D U L E L I S T S C H E M A S                      */
/*                                                                           */
/*****************************************************************************/
/* Gere la saisie des noms de fichiers et constitue la liste correspondante
*/

/*****************************************************************************/
/*                                  LISTSCHALLOC                             */
/*****************************************************************************/
/* Alloue la place necessaire a une structure de type lschelem */

HANDLE ListSchAlloc(hwnd)

HWND hwnd;
{
int longueur;
HANDLE hGlob;

longueur = sizeof(struct lschelem);
hGlob = GlobalAlloc(GHND,(long)longueur);
if (hGlob == NULL) MessageBox(hwnd,(LPSTR)"ALLOC RATEE",NULL,MB_OK);
return (hGlob);

}


/*****************************************************************************/
/*                                    ADDSCH                                 */
/*****************************************************************************/
/* Ajoute un element a listsch dont le nom est contenu dans schname et le nom
de fichier correspondant est contenu dans filename */

FAR PASCAL AddSch(hwnd,schname)

HWND hwnd;
char schname[50];

{
LISTESCH l, nouv;
HANDLE h;

h=ListSchAlloc(hwnd);
nouv=(LISTESCH)GlobalLock(h);

lstrcpy((LPSTR) nouv->filen,(LPSTR) filename);
lstrcpy((LPSTR) nouv->schn,(LPSTR) schname);

nouv->next=NULL;

if (listsch==NULL) listsch=nouv;
else
 {
  l=listsch;
  while (l->next != NULL)
        l=l->next;
  l->next=nouv;
 }
```

```
}

/*************************************************************************/
/*                              EDTPROC                                  */
/*************************************************************************/
/* Saisit le nom d'un fichier par l'intermediaire d'une EditBox */

BOOL FAR PASCAL Edtproc(hDlg, message, wParam, lParam)

HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;

{
 char schname[50];

 switch(message)
   {
    case WM_INITDIALOG:
         SetFocus(GetDlgItem(hDlg, ID_FILENAME));
         return (FALSE);
         break;
    case WM_COMMAND:
    switch(wParam)
      {
       case ID_OK:
            GetDlgItemText(hDlg, ID_FILENAME, (LPSTR)filename, 50);
            EndDialog(hDlg,0);
            lstrcpy((LPSTR)schname,(LPSTR)PosSch(filename));
            AddSch(hDlg,schname);
            return(TRUE);
            break;
       case ID_CANCEL:
            strcpy(filename, "9");
            EndDialog(hDlg,0);
            return(TRUE);
            break;
       default:
            return(FALSE);
            break;
      }
   default: return(FALSE);
   }
}

/*************************************************************************/
/*                            LISTSCHEMAS                                */
/*************************************************************************/
/* Gere l'appel a l'EditBox, la rappele jusqu'a ce que l'utilisateur
tape sur le bouton end, ce qui met 9 dans filename */

BOOL FAR PASCAL ListSchemas (hWnd)
HWND hWnd;

{
FARPROC lpproctemp;
char schname[50];
```

```
strcpy(filename, "A");
listsch=NULL;
while (*filename != '9')
  {
   lpproctemp=MakeProcInstance(Edtproc, hInst);
   DialogBox(hInst, (LPSTR)"EDT", hWnd, lpproctemp);
   FreeProcInstance((FARPROC)lpproctemp);
  }
}

/*****************************************************************************/
```